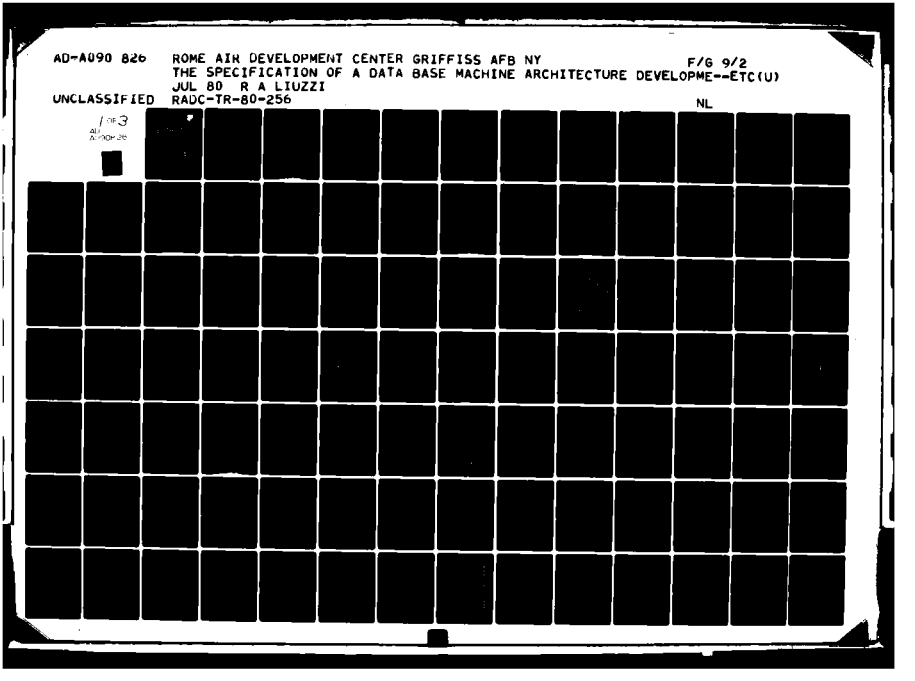
AD-A090 826 ROME AIR DEVELOPMENT CENTER GRIFFISS AFB NY F/G 9/2
THE SPECIFICATION OF A DATA BASE MACHINE ARCHITECTURE DEVELOPME--ETC(U)
JUL 80 R A LIUZZI
UNCLASSIFIED RADC-TR-80-256 NL



LEVEL II

RADC-TR-80-256
In-House Report
July 1980

AD A090826

THE SPECIFICATION OF A DATA BASE MACHINE ARCHITECTURE DEVELOPMENT FACILITY AND A METHODOLOGY FOR DEVELOPING SPECIAL PURPOSE FUNCTION ARCHITECTURES

RAYMOND A. LIUZZI

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
ELECTE
S OCT 28 1980 D
A

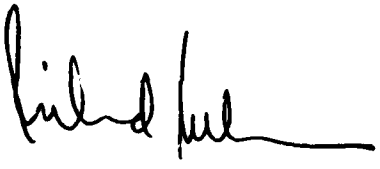
DDC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

80 10 27 023

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-80-256 has been reviewed and is approved for publication.

APPROVED: 

RICHARD NELSON
Chief, Information Processing Branch
Information Sciences Division

APPROVED: 

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: 

JOHN P. HUSS
Acting Chief, Plans Office

SUBJECT TO EXPORT CONTROL LAWS

This document contains information for manufacturing or using munitions of war. Export of the information contained herein, or release to foreign nationals within the United States, without first obtaining an export license, is a violation of the International Traffic in Arms Regulations. Such violation is subject to a penalty of up to 2 years imprisonment and a fine of $100,000 under 22 U.S.C 2778.

Include this notice with any reproduced portion of this document.

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIE) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-80-256	2. GOVT ACCESSION NO.: AD-A090826
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): THE SPECIFICATION OF A DATA BASE MACHINE ARCHITECTURE DEVELOPMENT FACILITY AND A METHODOLOGY FOR DEVELOPING SPECIAL PURPOSE FUNCTION ARCHITECTURES	5. TYPE OF REPORT & PERIOD COVERED: In-House
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Raymond A. Liuzzi	8. CONTRACT OR GRANT NUMBER(s): N/A
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Rome Air Development Center (ISIE) Griffiss AFB NY 13441	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 55811714
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIE) Griffiss AFB NY 13441	12. REPORT DATE: Jul 1980
	13. NUMBER OF PAGES: 290
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data Base Machine	Emulation	Computer Software
Data Management	Architecture	
Computers	Virtual Machines	
Microprogrammable Computers	Computer	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This research is concerned with specifying the set of components/tools needed in a Data Base Architecture Development (DMAD) Facility. A methodology is described to illustrate how this proposed facility can be used to develop special purpose function architectures (SPFAs). These SPFAs perform a data base management function in hardware that is currently performed in software on a sequential computer. This methodology includes a series of processes which are the select candidate function process

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

309050

and the create, test, evaluate, and substitute SPFA processes. Each process can be performed with a series of procedures that utilize tools/components of the DMAD facility.

The select candidate function process would use the facility to help choose candidate DBMS functions to be moved from software to hardware. For this process, assume that a data base management system (DBMS) executes on an emulated computer system in the facility. Software modules that comprise the DBMS and provide the DMBS functions would be studied as logical candidates to be moved into hardware. When a candidate is chosen one or more SPFAs may be chosen as replacements. This choice is made by developing these SPFAs as machines, using the facility and comparing their performance for a specific application.

A SPFA is originated when a new hardware architecture is proposed for a specific DBMS function. This SPFA can then be created in the DMAD facility by functionally describing the architecture with a hardware description language. This description, when compiled, produces a machine loadable version of the SPFA. This version, when configured in the DMAD facility, becomes a SPFA machine which is then available for execution in the facility. Once a SPFA machine is configured, it is available for testing, evaluation or substitution.

The testing process verifies that the SPFA machine performs its intended DBMS function. The evaluation process produces performance data for the SPFA machine based on timing its sequence of operations. Finally, the substitution process permits assessing hardware/software tradeoff issues needed to substitute the SPFA for a software DBMS module. This process permits a user to assess the impact of the evolutionary introduction of the SPFA into a current system.

A specific implementation of the DMAD facility is demonstrated using a QM-1 microprogrammable computer. The QM-1 is configured as separate merge machines based on two designs of merge processors. The first is Hollaar's merge processor (HOLL76) and the second is Stellhorn's (STEL76) inverted file processor. Each machine is described using the SMITE hardware description language.

Each machine is executed in the facility to demonstrate the feasibility of collecting various types of performance data. Timing data are generated for the SPFA machines based on timing their operations with an assumed set of hardware characteristics. These data are generated to illustrate that a specific set of characteristics can be studied to determine how they affect the performance of a SPFA machine. It is also shown how sensitivity measurements of a SPFA machine can be made for specific application criteria. Finally, modifications to the merge machines are made to demonstrate how the facility is used to tailor the development of a SPFA machine to meet the unique requirements of an application.

The tests and measurements conducted illustrate the feasibility of generating detailed analysis data prior to any actual hardware implementation of a SPFA. This type of data can be invaluable in helping project the highest qualified SPFA candidates to actually be hardware prototypes, and providing input that can be used in their actual hardware implementation.

TABLE OF CONTENTS

		Page
LIST OF FIGURES		i
LIST OF TABLES		iv
GLOSSARY OF TERMS		v
CHAPTER		
I.	INTRODUCTION	1
II.	REVIEW OF LITERATURE	6
	2.1 Introduction	6
	2.2 Architecture and Data Base Management	7
	2.3 Microprogrammable Computers	19
	2.4 Virtual Machine Technology	28
	2.5 Summary	33
III.	PROBLEM DEFINITION AND RESEARCH METHODOLOGY	34
	3.1 Introduction	34
	3.2 Problem Definition	34
	3.3 Research Methodology	37
	3.4 Summary	42
IV.	INTRODUCTION OF A DATA BASE MACHINE ARCHITECTURE DEVELOPMENT FACILITY	43
	4.1 Introduction	43
	4.2 Methodology to Develop a SPFA Using a DMAD Facility	44
	4.3 Architectural/Interfaces of the DMAD Facility	48
	4.4 Illustration of Utilization of DMAD Facility	50
	4.5 SPFA Development Process Functions	55
	4.5.1 Identifying Candidate Functions	55
	4.5.1.1 Select Candidate Process Functions	55
	4.5.2 Origination of a SPFA	56
	4.5.2.1 Create Process Functions	57
	4.5.3 Execution of a SPFA	57
	4.5.3.1 Testing Process Functions	58
	4.5.3.2 Evaluation Process Functions	61
	4.5.3.3 Substitution Process Functions	63

Page

4.6 Flow Within the DMAD Facility..... 64

4.6.1 Service Host Machine..... 64
4.6.2 Configuration Identification Machine..... 66
4.6.3 Master Configuration Control Machine..... 72
4.6.4 Data Base Function Execution Machine..... 74

4.7 Summary..... 77

V. DEVELOPMENT OF SPFA'S USING THE DMAD FACILITY..... 78

5.1 Introduction..... 78
5.2 Procedure for Select Candidate Function Process... 78

5.2.1 DBMS Machine Identification..... 78
5.2.2 DBMS Machine Execution..... 80
5.2.3 Candidate DBMS Function Selection..... 82
5.2.3.1 Data Collection..... 83

5.3 Procedure for Create Process..... 86

5.3.1 SPFA Description Development..... 86
5.3.2 SPFA Introduction..... 88

5.4 Service Host Machine..... 90

5.4.1 Input Request Control Module..... 92
5.4.2 Configuration Request Module..... 95
5.4.3 Initiate Resource Request Module..... 97

5.5 Configuration Identification Machine..... 99

5.5.1 Example SPFA Creation..... 101

5.6 Master Configuration Control Machine..... 104

5.6.1 Configuration Control Module..... 104
5.6.2 Request Module..... 106

5.7 Data Base Function Execution Machine..... 108

5.7.1 Input Control Module..... 108
5.7.2 Execution Control Module..... 110

5.8 Procedure for Test Process..... 112

5.8.1 Execution of SPFA Machine..... 112
5.8.2 SPFA Machine Functional Verification..... 114
5.8.3 SPFA Machine Debugging..... 117
5.8.4 Test Process Example..... 118

Page

5.9 Procedure for Evaluation Process..... 126

5.9.1 Identification of Hardware Operations.... 126
5.9.2 SPFA Machine Performance..... 132
5.9.3 Evaluation Process Example..... 134

5.10 Procedure for Substitution Process..... 139

5.10.1 DBMS/SPFA Identification..... 139
5.10.2 DBMS/SPFA Machine Selective Integration 142
5.10.2.1 Special Resource Requests..... 144
5.10.3 DBMS/SPFA Machine Analysis..... 147
5.10.4 Substitution Process Example..... 147

5.11 Summary..... 152

VI. TOOLS/COMPONENTS USED TO DEMONSTRATE A SPECIFIC IMPLEMENTATION OF DMAD FACILITY..... 153

6.1 Introduction..... 153
6.2 Organization of the Specific DMAD Facility..... 153
6.3 QM-1 Microprogrammable Compuer as Execution Machine..... 155

6.3.1 Use of QM-1 Main and Control Stores to Execute SPFA Machines..... 156

6.4 SMITE Hardware Description Language to Describe SPFA's..... 160

6.4.1 SMITE Features for SPFA Description.... 160
6.4.2 Timing Procedure for SPFA Machines..... 161
6.4.2.1 RTCA Assignment..... 163
6.4.2.2 RTCA Calculations..... 167
6.4.3 Identification of Operations Used for SPFA Machines..... 168

6.5 Summary..... 172

VII. DEMONSTRATION OF FEASIBILITY OF UTILIZING A SPECIFIC DMAD FACILITY TO DEVELOP SPECIAL PURPOSE FUNCTION ARCHITECTURES..... 173

7.1 Introduction..... 173
7.2 Use of Facility To Establish MERGE As Candidate DBMS Function..... 173
7.3 Initiate Development of MERGE SPFA's Using Facility..... 183

7.3.1 Create Hollaar's MERGE Processor..... 183
7.3.2 Stellhorn's Inverted File Processor..... 184

Page

7.4 Execution of MERGE Machines in the Facility..... 191
7.5 Use of Facility to Evaluate MERGE Machines..... 197

7.5.1 Sample Timing Calculations..... 201
7.5.2 Evaluation Results..... 207
7.5.2.1 MERG1 Machine..... 207
7.5.2.2 MERG2 Machine..... 216
7.5.2.3 Additional Evaluations..... 219
7.5.2.4 MERG3 Machine..... 219

7.6 Other Comparisons..... 226
7.7 Summary..... 229

VIII. DISCUSSION AND FUTURE RESEARCH..... 230

8.1 Discussion..... 230
8.2 Future Research..... 235

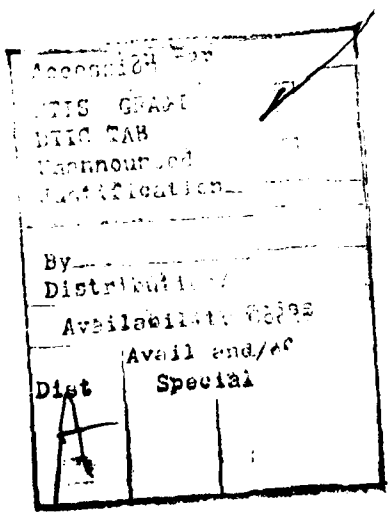
APPENDIX A..... 239

APPENDIX B..... 247

APPENDIX C..... 259

BIBLIOGRAPHY..... 266

BIOGRAPHICAL DATA..... 273



LIST OF FIGURES

Figure		Page
2-1	Areas of Research Conentration	8
2-2	Data Base Machine Evolution	10
2-3	Associative Processor Architecture	14
2-4	Typical Instruction Execution Sequence	20
2-5	Illustration of a Virtual Machine Monitor	29
2-6	Illustration of Virtual Peripherals	31
3-1	Research Methodology	39
4-1	Development Processes	45
4-2	Development Process Flow Model	47
4-3	Architectural/Interfaces of the DMAD Facility	49
4-4	Developing a Special Purpose Function Architecture	52
4-5	Substitution of a Special Purpose Function Architecture	54
4-6	Illustration of Transformation of DMAD Facility	59
4-7	Flow Within Service Host Machine	65
4-8	Flow Within Configuration Identification Machine	67
4-9	Illustration of a Data Base Machine Architecture Configuration Array (DMCA)	68
4-10	Illustration of a Realization Timing Control Array (RTCA)	71
4-11	Flow Within Master Configuration Control Machine (MCCM)	73
4-12	Flow Within Data Base Function Execution Machine (DBFEM)	75
4-13	Illustration of Execution Machine Types	76
5-1	Illustration of DBMS Machine Execution	79
5-2	DBMS Machine Identification Using a Data Base Machine Architecture Configuration Array	81
5-3	Procedure to Identify DBMS Functions Using DMAD Facility	84
5-4	DMAD Facility Environment for Developing Machines	87
5-5	SPFA Machine Identification Using a Data Base Machine Architecture Configuration Array	89
5-6	Flow Within Service Host Machine Input Request Control Module	91
5-7	Sample Format of Input to Configuration Identification Machine	94
5-8	Flow Within Service Host Machine Configuration Request Module	96
5-9	Flow Within Service Host Machine Initiate Resource Request Module	98
5-10	Flow Within Configuration Identification Machine	100
5-11	Example Creation of a SPFA Machine	102
5-12	Flow Within Master Configuration Control Machine	105
5-13	Sample Format of a Request Block	107
5-14	Flow Within Data Base Function Execution Machine	109
5-15	Configuration of Machine Type on DBFEM	111
5-16	Execution of a SPFA Machine	113

Page

Figure

5-17 A Verification Procedure for a SPFA Machine..... 115
5-18 Example SPFA Machine Identification..... 120
5-19 Sample Test Process Query..... 121
5-20 Configuration Request Block..... 122
5-21 Execution Request Block..... 124
5-22 Configuration of SPFA Machine for a Test Process..... 125
5-23 Types of Performance Considerations for SPFA Operations 128
5-24 Entries in a Realization Timing Control Array (RTCA)... 131
5-25 Example Realization Timing Control Array Utilization... 133
5-26 Generation of Timing Assignments Using a Realization Timing Control Array..... 136
5-27 Configuration of SPFA Machine for Evaluation Process... 137
5-28 Sample Format of a Function Trap Instruction..... 141
5-29 Illustration of a Virtual Data Base Machine Monitor.... 143
5-30 Sample Types of Special Resource Requests..... 145
5-31 DBMS/SPFA Machine Identification Using a Data Base Machine Architecture Configuration Array (DMCA)..... 149
5-32 Example of DMAD Facility Transformed into a DBMS/SPFA Machine..... 151
6-1 Components Utilized for Specific DMAD Facility Demonstration..... 154
6-2 Illustration of Architecture of QM-1 Microprogrammable Computer..... 157
6-3 Utilization of QM-1 Two Level Control Store Hierarchy for SPFA Machines..... 158
6-4 Outline of SPFA Realization Timing Control Array..... 162
6-5 A Comparison Operation for a SPFA Machine..... 165
6-6 Example of an Information Transfer Operation..... 169
7-1 Merge SPFA Development Using a QM-1 Microprogrammable Computer..... 174
7-2 Configuration of Facility to Collect Utilization Data.. 176
7-3 Illustration of Procedures to Collect DBMS Utilization Data Using Facility..... 177
7-4 Merge Software Module Interfaces..... 179
7-5 Hollaar's Merge Processor as "MERG1" SPFA..... 185
7-6 "MERG1" SPFA Operations for Compare Processor..... 186
7-7 Stellhorn Inverted File Processor as "MERG2" SPFA..... 188
7-8 2 X 2 MERG2 Network Used For Compare Processor for MERG2 SPFA..... 189
7-9 Configuration of "MERG1" or QM-1..... 192
7-10 Use of Configuration Files for MERGE Machines..... 193
7-11 Sample Input Driver for MERG1 SPFA..... 195
7-12 Illustration of a MERGE Machine on QM-1 Microprogrammable Computer..... 196
7-13 MERGE SPFA Machine Operations..... 198
7-14 Time for Operations Required by SPFA Machines..... 203
7-15 MERG1 Realization Timing Control Array..... 205
7-16 MERG2 Realization Timing Control Array..... 206

Figure

Page

7-17 Results of One Cycle of MERG1 Machine for Realization Assumption 2..... 209
7-18 Timing Data For MERGI Machine..... 210
7-19 Sample Input Lists Configured for MERG1 Machine..... 211
7-20 MERG1 Machine Output for "OR" MERGE Type Request..... 212
7-21 Timing Results for MERG1 Machine for "OR" MERGE Type Request..... 213
7-22 Comparison of Timings for Merge Machines Using Facility. 215
7-23 Timing Results for MERG2 Machine for "OR" MERGE Type Request..... 217
7-24 Illustration of Using Facility to Study MERGE Operation Types..... 220
7-25 Architectural/Interfaces for MERG3 SPFA..... 222
7-26 Results of Comparing MERG1 and MERG3 Machines for Same MERGE Operations..... 224

LIST OF TABLES

Table		Page
5-1	Characteristics of Semi-Conductor Technologies.....	129
7-1	Performance Characteristics of MERGE Function Using Various MERGE Methods.....	226

GLOSSARY OF TERMS

APU:	Associative Processor Unit
ARM:	Active Request Module (MCCM)
BRM:	Build Request Module (MCCM)
CCM:	Configuration Control Module (MCCM)
CIM:	Configuration Identification Machine
CM:	Control Module
CMM:	Configuration Manager Module (MCCM)
CR:	Comparand Register
CRB:	Configuration Request Block
CRM:	Configuration Request Module (SHM)
CRDM:	Configuration Request Decoder Module (SHM)
CU:	Control Unit
DBFEM:	Data Base Function Execution Machine
DBM:	Data Base Machine
DBMA:	Data Base Machine Architect
DBMS:	Data Base Management System
DBMS/SPFA Machine:	Data Base Management System/Special Purpose Function Architecture Machine
DIM:	Device Interface Module (DBFEM)
DMAD Facility:	Data Base Machine Architecture Development Facility
DMCA:	Data Base Machine Configuration Array
ECM:	Execution Control Module (DBFEM)
ECR:	Execution Configuration Request
ERB:	Execute Request Block
FTI:	Function Trap Instruction

GCRM: Generate Configuration Request Module (SHM)

GRRM: Generate Resource Request Module (SHM)

HDL: Hardware Description Language

ICM: Input Control Module (DBFEM)

IDM: Input Decoder Module (MCCM)

IRCM: Input Request Control Module (SHM)

IRRM: Initiate Resource Request Module (SHM)

LCU: Local Control Unit

Mpe_{sw}: Metric for Processing Efficiency of a Software Module

MCCM: Master Configuration Control Machine

MCM: Machine Configuration Module (DBFEM)

MR: Mask Register

OGn: Operation Group n

OGn(s:s+r): Operation Group Consisting of Statements s to s + r

PC: Program Counter

PFM: Process Flow Model

PM: Pre-processor Module (SHM)

PDL: Process Description Language

QFM: Query Formulation Module (SHM)

RA: Realization Assumption

RCR: Resource Configuration Request

RM: Request Module (MCCM)

RR: Request Register

RRB: Resource Request Block

RRM: Resource Request Module (SHM)

RTCA: Realization Timing Control Array

SAP: Software Application Program

SHM: Service Host Machine

SMTM: Select Machine Type Module (DBFEM)

SPFA: Special Purpose Function Architecture

SPFA/IG: Special Purpose Function Architecture/Input Generator

SRRM: Special Resource Request Module (DBFEM)

TIMER-VALUE (n): TIMER-VALUE assigned to operation n in SPFA description

$t_{compare}$: time for compare operation

t_{IC}: time for selected integrated circuit

$t_{ind\ set}$: time for indicator set operation

t_{select}: time for select operation

t_{shift}: time for shift operation

$t_{transfer}$: time for transfer operation

$t_{(n,m)}$: time for operation n using realization assumption m

$t_{(O,s,m)}$: time for operation O, at statement s, using realization assumption m

$t_{(OG,s:s+r,m)}$: time for operation group OG, at statements s to s + r for realization assumption m

VDMM: Virtual Data Base Machine Monitor

VMM: Virtual Machine Monitor

CHAPTER I

INTRODUCTION

Interest in computer architecture research, as applied to data base management, has recently increased because of the advancing state-of-the-art in inexpensive, fast, and new hardware components. Hardware technology advances are occurring primarily in three areas: central processing units (CPU), semi-conductor random-access memory (RAM), and all-electronic bulk memories. The cost-to-performance ratio of CPU's will decline rapidly over the next ten years. Low cost CPU's with the performance capabilities of today's medium-priced minicomputers will be available for hundreds of dollars in the 1980's. Large and faster associative processors will be introduced (BERR79). New technologies in memories utilizing bubbles or charge couple devices will replace existing fixed head discs.

These advances have prompted organizations to become increasingly aware of the effect new hardware architectures will have on data base management system (DBMS) performance. In the past, data base management functions have been performed in software on sequential computers. However, research efforts have demonstrated that the performance of some of these functions can be increased substantially on different computer architectures (SING77,CAPR78,FREE78). Most notably, concurrency and content addressable architectural features favorably increase the DBMS searching function for specific data base applications. This has resulted in an increased interest in the concept of a Data Base Machine (DBM).

These machines perform, in hardware, various data base management functions. The decreasing cost of hardware has made it feasible to examine separately designed data base machines to support a specific data base management problem.

The characteristics that comprise a data base machine have been defined by Liuzzi and Berra (LIUZ78). Part of this definition states that a data base machine can be composed of a set of specialized architectures called special purpose function architectures (SPFA's). Each SPFA is defined as a hardware architecture that performs a specific data base management function. The choice of a SPFA, for inclusion with a specific data base management system, depends on several considerations:

(1) what SPFA should be chosen to replace a software DBMS function, if more than one exists,

(2) does the performance improvement resulting from the SPFA warrant change from software to hardware,

(3) what are the hardware implementation strategies of the SPFA that insure that its cost, performance, and reliability factors are optimized, and

(4) how well can the SPFA be integrated within a DBMS to insure that it can help preserve DBMS software application programming investments?

The considerations cited above must be examined if an organization decides to expand its data base management capability by introducing SPFA's, and as more and more SPFA's appear, this choice will become more difficult. Thus, procedures are needed to help optimize the choice of a

SPFA for a specific DBMS application. The research described in this report concerns the proposed development of a facility, which consists of an integrated set of tools/components that can help a user make this choice.

Several research developments have been integrated into the design of this facility. These research areas are described in CHAPTER II and consist of:

(1) data base management and architecture,

(2) microprogrammable computers and emulation, and

(3) virtual machine technology.

A research methodology is described in CHAPTER III that combines selected activities in each of the above areas into the specification of a set of tools, components and procedures that comprise this facility. Described in CHAPTER IV is the overall specification of the Data Base Machine Architecture Development (DMAD) Facility. This specification includes identifying the architectural/interfaces of hardware components in the facility. These components are integrated to provide a capability for user's to develop DBMS special purpose function architectures. A methodology is then described to utilize the facility with a series of processes which are select candidate function, create, test, evaluate and substitute.

The select candidate function process helps choose candidate DBMS functions to be moved from software to hardware. The create process is used to functionally describe architectures that can perform the DBMS function. The test process verifys that each SPFA performs its intended DBMS function. The evaluation process is used to generate data concerning each SPFA's performance. Finally, the substitution process permits an

assessment of tradeoffs necessary to integrate a SPFA with a given DBMS.

Described in CHAPTER V is a detailed examination of the above processes. Each process can be performed with a series of procedures that utilize tools/components of the facility. First, the facility is used to emulate existing computing systems to help choose candidate DBMS software functions. Next, several SPFA's that can perform this function are examined. Each SPFA created becomes a machine in the DMAD Facility. This machine can now be executed, tested, and evaluated using a unique set of tools and procedures available to a user. Its development is based on optimizing its design, description and hardware implementation characteristics to achieve optimal performance for a specific DBMS application.

Described in CHAPTER VI is the set of specialized tools/components that are utilized to demonstrate a specific implementation of the DMAD Facility. A Nanodata QM-1 microprogrammable computer is used as a vehicle to execute the SPFA machine in this facility.

Described in CHAPTER VII is a demonstration of how this specific facility is used to develop SPFA's. This demonstration consists of illustrating how the facility is used to select the merge DBMS function as a candidate to be moved from software to hardware and actual use of the specific facility implementation to develop several merge SPFA's. One is based on Hollaar's (HOLL76) merge processor and another is based on Stellhorn's (STEL77) inverted file processor. Each performs the same merge DBMS function using different merge algorithms. Both merge SPFA's are executed within the facility as merge machines. Tools/components within the specific facility are used to create, test and evaluate

each merge SPFA. Several types of evaluation are conducted to illustrate various capabilities of the facility for architecture evaluation.

It is also shown in CHAPTER VII that additional performance comparisons can be made using the facility for other approaches to perform the same DBMS function. These comparisons are made in a common environment to demonstrate how a DMAD Facility can be used to cite advantages and disadvantages of alternative approaches for performing data base management functions in software, firmware and hardware.

Described in CHAPTER VIII are several results and discussions of this research. A number areas for further research are also described.

CHAPTER II

REVIEW OF LITERATURE

2.1 INTRODUCTION

Current activities in data base management and computer architecture have motivated the interest in developing data base machine architectures as described in this report. Interest in developing these architectures represents a study of the migration of DBMS functions from software to hardware. Just how this migration will occur and how data base machines will become a reality is postulated by Berra (BERR78).

Berra states that data base machines will be developed by both revolutionary and evolutionary approaches. Practitioners of the revolutionary approach will bypass much of the current investment in data base management systems and choose to go directly to the new data base machine architecture. This approach will have many problems and failures but will also provide many of the leading technological advances. Berra (BERR78) further states that organizations following the evolutionary approach will likely bridge the gap between current data base management systems and data base machines by introducing new technologies at minimal cost and service interruptions. In this approach data base machines will evolve as new functions and architectures are selectively added to existing systems.

In order to help organizations develop data base machines, using the evolutionary approach, the following areas have been examined:

(1) data base management and architecture,

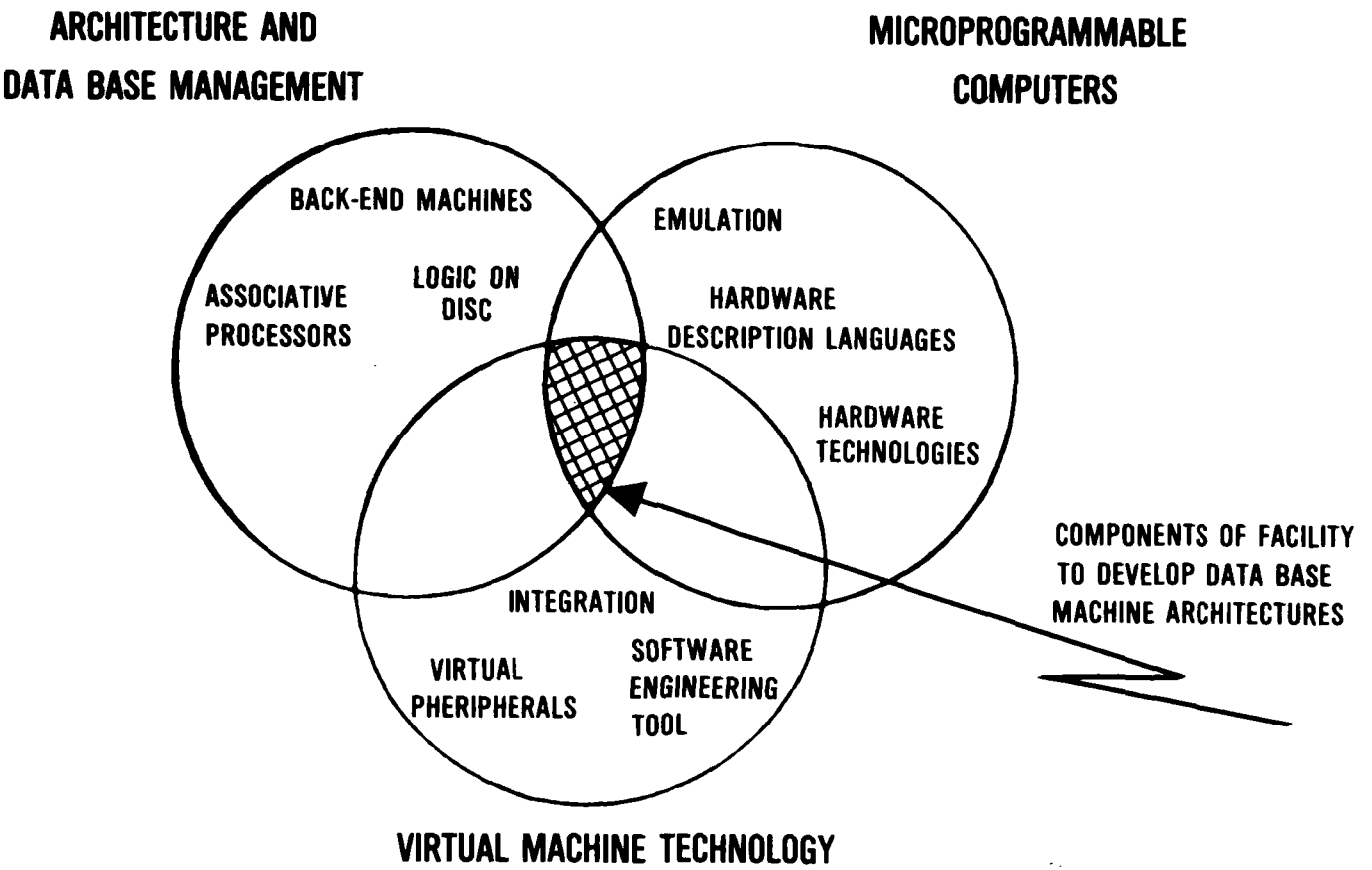
(2) microprogrammable computers, and

(3) virtual machine technology.

Conceptually illustrated in Figure 2-1 are each of these research areas. Several tools, procedures and technology from efforts in these areas have been combined to define a facility which can help develop data base machine architectures. Described in the next three sections of this chapter are specific research tasks from each of these areas that contribute to the specification of this facility.

2.2 ARCHITECTURES AND DATA BASE MANAGEMENT

The effects of architecture on data base management have been far reaching over the past decade. New techniques and technologies are emerging which can provide more efficient ways to manipulate data. These technologies make it feasible to examine ways to improve DBMS performance. One procedure is to examine new architectures that can perform DBMS functions currently done in software. Many of these software functions absorb a great percentage of a DBMS's computing power. An often proposed solution to this problem is to remove some of the time consuming functions from software and perform them in hardware. How to determine which functions to implement in hardware, and how to choose their optimal architecture, for a given user application, becomes a very difficult task. In order to help ease this transition, data base machines have been introduced as new hardware architectures designed to perform specific DBMS functions (CHAM79).



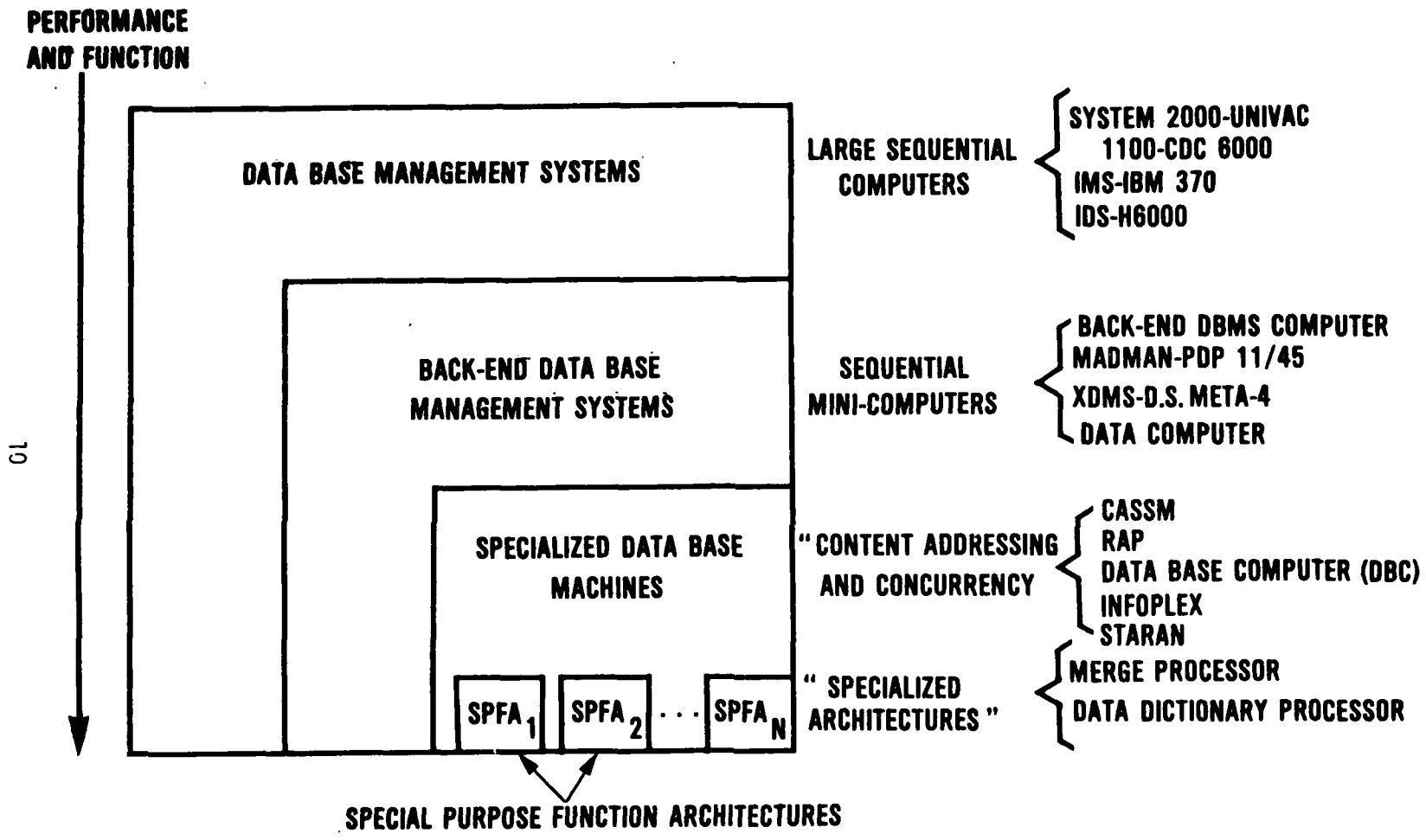
AREAS OF RESEARCH CONCENTRATION

FIGURE 2-1

The notion of a Data Base Machine (DBM) has evolved primarily because of the need to accomplish data base management tasks more efficiently. The evolution to the current class of data base machines can be traced by viewing Figure 2-2. Data base management systems (DBMS) were originally developed to execute on large sequential systems and had to rely on the services of a generalized host execution system to perform many of its tasks. Examples of this class of system includes the IDS system on an H6000, IMS on an IBM 360/270, and System 2000 on several large machines (KOEM73). However, much of the processing efficiency of these systems is compromised by the inefficiency of I/O operations for processing data. The operating systems on these large machines must multi-task a number of activities.

As minicomputer development progressed, it became apparent that many data base management tasks could be accomplished more efficiently by removing them from the large sequential machine to a machine dedicated entirely to data base management. Such a machine is called a back-end machine. Canady, et al outlined an architecture for performing various DBMS tasks on a back-end Digital Scientific Meta-4 Computer (CANA74). Numerous advantages were cited including security, reliability, and efficiency. These notions motivated interest in developing data base management systems on minicomputers. For instance, the MADMAN System was developed by General Electric to execute on PDP 11/45. A recent effort has implemented the MADMAN System on a Honeywell H6000 (PHIL78).

Further development in semiconductor technologies has produced the microprocessor and microcomputer, along with the notion that it



DATA BASE MACHINE EVOLUTION

FIGURE 2-2

is economically feasible to develop computers which are primarily designed for data base management. Previous research efforts have substantiated the fact that computer architectures that provide concurrency and content addressing constructs can provide order of magnitude increases in the performance of certain data base management functions (MOLD73,SING77). As a result, a new class of machines have emerged with various unique architectures designed to provide these constructs. They are referred to as data base machines. Liuzzi and Berra (LIUZ78) have defined a set of characteristics for a range of data base machines that consist of:

(1) an overall architecture composed of one or more special purpose function architectures (SPFA'),

(2) an architecture that is based on parallelism and content addressing,

(3) a set of compatible memory units for the storing and efficient retrieval of data, and

(4) an architecture which is a back-end machine.

Several types of machines have been reported in the literature with these characteristics. They include:

(1) the logic per track architecture of the University of Florida's Context Addressed Segment Sequential Memory Organization (CASSM) System (COPE75),the multicell CASSM system (SU79), and the University of Toronto's Relational Associative Processor (RAP) System (SHU75,SHU79). These systems attain concurrency by moving logic from the central processing unit to the individual disk heads that read data from each tract on a fixed head disk. The RAP System has recently been extended to include semiconductor charge couple device (CCD), random access memory (RAM) or bubble memory.

(2) the Ohio State Data Base Computer (DBC) proposed by Hsiao (HSIA77,BANE79) and consists of a unique architecture that interconnects several specialized processors aimed at supporting secure large-scale data bases. Each data base is stored on content-addressable moving head disk devices. Emerging technologies such as magnetic bubbles and CCD's have been chosen for the structure memory.

(3) the INFOPLEX System proposed by Madnick (MADN78) which takes advantage of new memory and processor technologies to organize a "smart" memory hierarchy to handle the storing and retrieval of information. Its information management functions are decomposed into a functional hierarchy implemented by a hierarchy of microprocessors.

(4) the DIRECT System proposed by DeWitt (DEWI79) which is a multiprocessor organization for supporting relational data base management systems. DIRECT has a multiple-instruction, multiple-data stream architecture. It can simultaneously support both inter-query and intra-query concurrency.

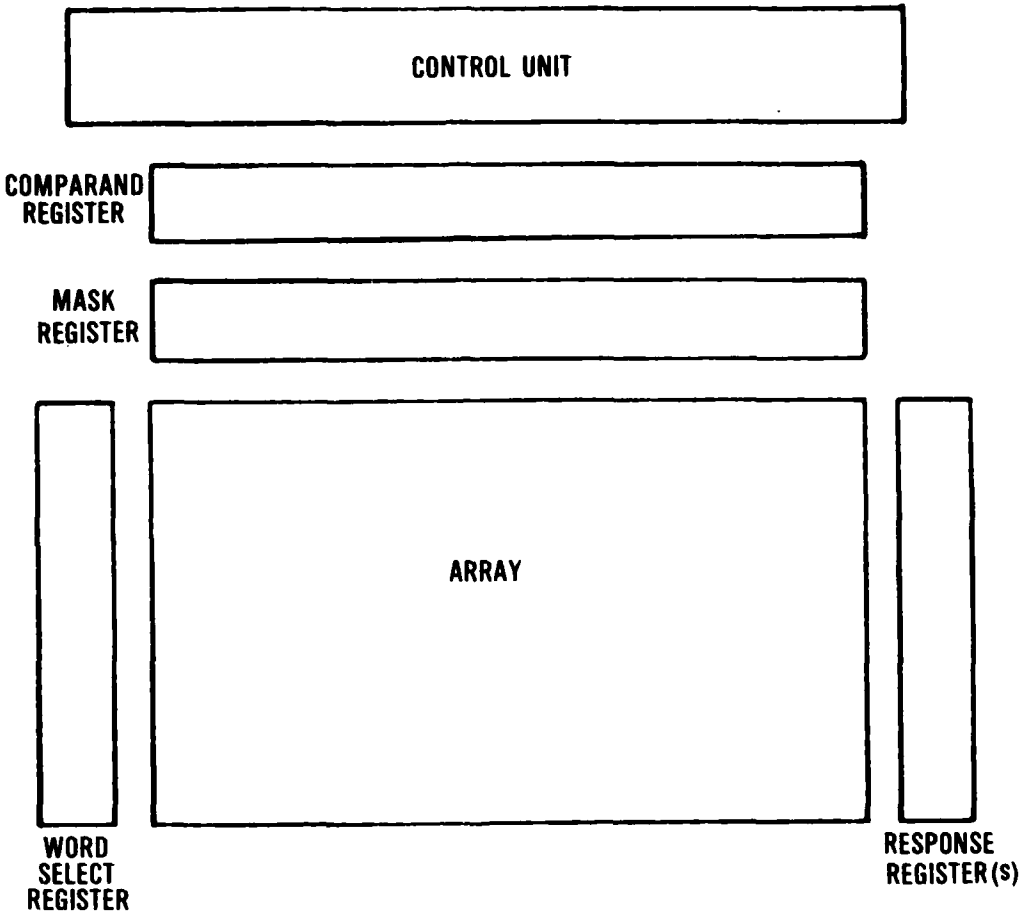
(5) the associative processor which has been experimentally examined for data base management applications. Early studies by DeFiore, Stillman, and Berra (DEFI73,STIL74) using the Goodyear Associative Memory established that searching a data base was significantly improved by associative techniques. This has led to a series of studies between Syracuse University and RADC utilizing a Goodyear Staran B Associative Array Processor (FELD73). These studies have led directly to an effort to develop a pilot data base management capability on this processor (RADC78).

The associative processor is unique in the area of data base ma- machines because of the type of capabilities it possesses. Basically,

associative processors are distinguished from conventional computer memories by the fact that they operate on arrays of data addressed by tags or value rather than by location. Illustrated in Figure 2-3 is a typical architectural organization of an associative processor. The array is the storage device which contains the data to be searched. The comparand register contains the values to be processed against the data in a search operation. The mask register enables or inhibits bit slices from participating in a given operation. The response register records the search results and permits the Boolean operations to take place through logical combinations of the registers.

Several studies of this architecture compared to a conventional serial processing architecture have indicated that associative processors could easily be suitable for data base management because of their rapid searching capabilities. The RELACS System proposed by Oliver (OLIV79) is a data base management system that utilizes associative processors to implement the relational data model. RELACS is capable of supporting many functions of a data base management system including retrieve, updating, deletion, modification, and addition.

Many of the systems described have been designed to take advantage of several new storage devices that currently exist for the efficient retrieval of data. For instance, current studies are being conducted for interfacing the STARAN processor to a large mass memory unit (RADC78). Several other memory technologies including bubble memories and charge coupled devices (CCD's) are being considered for storing data and may provide the needed breakthrough for the efficient storing and searching of large data bases (HSIA78).



ASSOCIATIVE PROCESSOR ARCHITECTURE

FIGURE2-3

In addition to this class of DBM's, a set of specialized architectures have emerged. As Figure 2-2 indicates, these specialized architectures can form portions of a data base machine. They are primarily designed to optimize a single data base management function. Several different types of data base functions have been designed as specialized architectures.

Lang and Nahourali (LANG76) have proposed an architecture which extends third generation computer systems by adding commercially available microprocessors to aid in the searching of a direct access storage device. Their studies indicate this additional architectural capability can improve data base performance in large data base applications.

Roberts (ROBE76) has proposed a specialized parallel computer architecture for high speed searching of large textual files. The data base to be searched is partitioned among independent high speed serial-access memories that are searched in parallel by dedicated microprocessors connected to a common communication bus.

Hollaar (HOLL76) has proposed a specialized merge processor that combines data from sorted input lists into a sorted output file. This processor is designed with architectural constructs that form the merge operation. Stellhorn (STEL77) proposed an inverted file processor that utilizes a specialized architecture to access files of document identifiers and perform the processing associated with a Boolean search request. Hollaar and Stellhorn (HOLL76) propose a specialized architecture for textural information retrieval. The basic architecture of the system consists of several parallel search modules connected to a disc via a parallel/serial interface. This architecture is especially

suited for list merging updating and sorting operations. Hollaar (HOLL79) has also extended this work to include the design of a list merging network.

Ramamoorthy et. al., (RAMA) have proposed a fast sorting associative memory. This SPFA is a bit machine that implements search operations parallel by word and serial by bit slice.

Mukhopadhyay (MUKH79) has proposed specialized hardware algorithms for non-numeric computation. These algorithms can be implemented using various LSI technology for high speed pattern matching needs.

Love (LOVE73) has proposed an architecture that allows a series of associative processors to operate as a single large associative memory, or to operate individually in concurrent independent operations. Capraro (CAPR79) has proposed to integrate a data dictionary in a specialized hardware architecture that utilizes associative processors. Berra and Singhania (SING76) have designed a special purpose function architecture utilizing associative memories for pipelining a directory to a very large data base. The basic notion of this architecture is that real-time requirements dictate the utilization of a directory to the data base. The results of this study indicated that the pipeline system provides faster retrieval than sequential inverted list systems, especially in the case of multiple-key retrievals. Karlowshy and Leilich (KARL75) from the Technical University of Braunschweig have proposed a special purpose function architecture called a "search" processor to search data stored on mass memory without using the CPU and I/O of a host computer.

Each of these special purpose function architectures can replace

a DBMS function that is currently being performed in software on a sequential machine. However, it is not clear which SPFA an organization should choose or how its development can be optimized for a specific application. Each SPFA should be examined thoroughly, prior to actual development, to determine its effectiveness for the application.

2.2.1 Data Base Machine Architecture Performance Evaluation

Several methods exist to evaluate the performance of data base machine architectures. These include:

(a) mathematical modeling,

(b) simulation, and

(c) emulation.

The data base machine architect can utilize mathematical modeling techniques to derive timing equations that can be used to evaluate the performance of proposed architectures (SING76,OLIV79,CAPR78). The general approach is to first develop timing data for various microfunctions. These data can be obtained from various handbooks on circuit devices or from published literature concerning the performance of these microfunctions on existing computers (i.e. STARAN). The next step is to develop macrofunctions which are composed of one or more microfunctions and to develop timing data for these. Finally, one combines the macrofunctions into jobs that are performed in a data base management context (i.e. update, query, etc.) and thus obtains performance data.

In other cases, a stochastic analysis can be used to postulate an underlying stochastic process that governs the behavior sequences on the performance of a DBMS architecture in finite observation periods.

Both the timing equation approach and the stochastic modeling approach fall in the general category of analytical modeling. An alternative is operational analysis (SERV79) based on using measurable values, testable assumptions and finite observation periods, but are only possible on systems already in existence.

Simulation is the process of designing a model of a real or proposed system and conducting experiments with this model for the purpose of either understanding the behavior of the system or of evaluating various strategies for the operation of the system. The simulation of data base machine architectures may be quite complex and require a digital computer in order to perform the experiments and gather the data. A simulation would likely be used if one is unable to utilize an analytical modeling approach.

An emulation is a very specialized kind of simulation characterized by the following properties:

(1) in an emulation, both the simulator and "simuland" are computers, and

(2) in an emulation one comes much closer to reality by focusing attention on the behavior of the architecture at the instruction level rather than at the input-output level.

Thus, a simulation can be conducted using a general host computer whereas a specialized host computer with features such as microprogrammability are required to conduct an emulation.

Emulation, as defined by Mallach (MALL78), involves building or modifying a computer system to execute programs written for another system, in addition to its own programs. With a tool known as an

"emulator", a user can define a new machine on a system. Emulations have proven to be powerful tools for certain software conversion problems. During an upgrading process a user may define an emulator of an older system on a new machine and use the emulator to initially run old programs. Once this type of compatibility is established, the user may then choose to convert the programs to the native mode of the new system.

Other uses of emulators have been to develop special purpose computer systems for unique environments. These specialized architectures can be developed without incurring the expense of developing a one-of-a-kind hardware unit for this application.

The emergence of new microprogrammable computers provides the feasibility of utilizing emulation as a tool to examine new hardware architectures. These computers enable a wider range of architectures to be emulated and provide the ability to examine, in more detail, the actual hardware architecture being developed. Several architectural considerations are included in a microprogrammable computer that enable it to emulate a variety of architectures. These considerations are discussed in section 2.3.

2.3 MICROPROGRAMMABLE COMPUTERS

The development of generalized microprogrammable computers has enabled computer architects to develop emulations of several computer architectures. These emulations are produced by a series of microprograms that execute on a microprogrammable computer.

2.3.1 Microprogramming

In a simple sense, microprogramming may be considered an alternative attempt for implementing the architecture of a computer. Every programmable device possesses an architecture and an instruction set.

The building blocks for a digital computer consist of a central processing unit, which performs arithmetic and logic operations, a main store memory unit which stores intructions and data, input/output units which communicate with peripherals, and a control unit which sequences other components through various states while executing instructions. Shown in Figure 2-4 are the three phases of an instruction sequence which include:

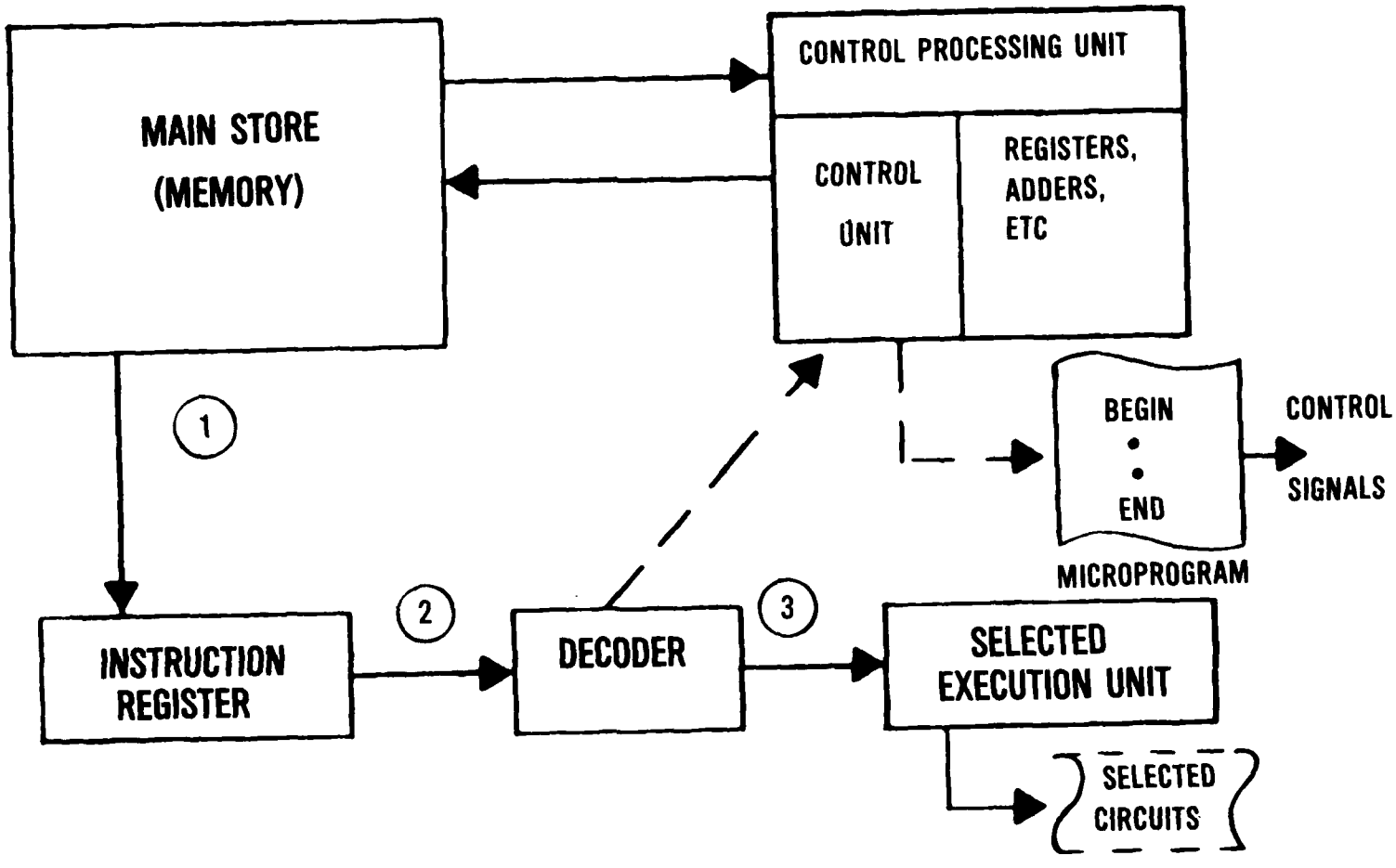
(1) instruction fetch,

(2) instruction decode, and

(3) instruction execution.

Instruction execution is initialized by fetching a machine instruction from a given memory location and placing it into an instruction register (1). The operation code of the instruction is transferred to a decoder (2). Once decoded, an execution plan (3) is determined such as data fetch, data manipulation, or data store. In the conventional or hardwired computer, a hardware decoding of the relevant portion of the instruction word selects one of several logic circuits, each of which is responsible for generating and sequencing the primitive signals of a given machine instruction.

Replacing this predetermined hardwired control unit with a more flexible computerized control unit (with its own control store memory) yields a microprogrammable computer. Now, following the instruction decode operation, the instruction opcode is used to point to a microprogram in control store which has been written to provide the control signals for that instruction. Hence, the functional nature of the computer as seen by the programmer is predetermined by the micro-



TYPICAL INSTRUCTION EXECUTION SEQUENCE

FIGURE 2-4

programmer and may be redefined as readily as the control store may be reprogrammed.

The ability to write into these control stores allows the microprogram machine to alter the interpretation of the instruction and hence by altering the microprogram one can alter the design of the architecture it is emulating. This feature is extremely important when considering the various architectural options available for data base machine architecture. Several features needed in a microprogrammable computer to emulate a variety of architectures are discussed in the next section.

2.3.2 Microprogrammable Computer Hardware

The basic functional components of a microprogrammable computer are main memory, control store, arithmetic and logic units, and the interconnections among these units. The design of each of these units can vary among several microprogrammable computers.

For instance, the control store of microprogrammable computer is subject to a number of variables including structure, size, speed, and loading conventions. Several variations in control store structures are cited by Agrawala (AGRA76). One example is the two level control store structure where micro-instructions in the lower level storage unit interpret micro-instructions in the upper level storage unit. This provides, to a user, added flexibility in the overall design of an emulator.

The size of control store depends on the number of bits in a word and the number of words in control store. The number of bits in a micro-instruction depends on the number of operations the

computer can perform and the parallelism involved in executing these operations. The speed of control store is generally affected by cost and speed of main memory. These two must be balanced to insure cohesive operation.

The architecture of certain microprogrammable computers allows the choice between one of two formats for the micro-instructions used in an emulation. These are vertical and horizontal micro-instructions.

Vertical instructions generally represent single micro-operations. They resemble conventional machine language instructions. They can specify a time sequence of control signals which allows them to be executed less frequently. In contrast, horizontal instructions represent several micro-operations that are executed concurrently. Because they control multiple resources concurrently, they contain more information than vertical instructions and have a greater length. Illustrated below are examples of currently available microprogrammable computers and types of computer architectures that have been emulated on these computers.

MICROPROGRAMMABLE COMPUTER	COMPUTERS EMULATED
NANODATA QM-1	PDP 11/0 NOVA 1200 S/360 MODELS 40,50 IBM 7090 CDC 1604A
BURROUGHS B-1700	IBM 1401 IBM 1440 IBM 1460 B-100 B-500

2.3.3 Hardware Description Languages

Hardware Description Languages (HDL) have been developed to help develop emulations of computer systems. These languages can provide several advantages, as cited by Chu (CHU74), which are:

(1) they serve as a means of communication,

(2) they permit a precise, yet concise, description,

(3) they provide convenient documentation,

(4) they are amendable to simulation on a computer, and

(5) they aid in an integrated design and automation.

An HDL can assist computer architecture development at various levels of abstraction for a computer system. These levels include:

(a) the device transfer level,

(b) the register transfer level, and

(c) the gate/flip/flop level.

The computer architect chooses the level of development by the HDL chosen to describe an architecture. Several different languages exist that can help in describing an architecture at each of these levels. These languages include:

(1) A Hardware Programming Language (AHPL) introduced by Hill (HILL74) at the University of Arizona, that is a hardware description language based on the notational conventions of APL.

(2) Computer Design Language (CDL) based on the work by Chu (CHU74) at the University of Maryland. This language describes the computer elements and hardware algorithms at a level just above that of the electronics.

(3) Digital System Design Language (DDL), based on the work of Dietmeyer (DIET74) at the University of Wisconsin, is a block oriented language. The blocks of a description are intended to correspond to the sub-systems of the hardware system.

(4) Logic Algorithm Language (LOGAL) based on Stewart's Work (STEW77) at Univac. LOGAL permits design of the logic design algorithms and iterates their design trade-offs for optimization to the logic designer.

(5) Instruction Set Processor (ISP) based on Siewiorek's work (SIEW74) at Carnegie-Mellon University. ISP was specified by Bell and Newell to permit description of the programming level of a computer and has recently been utilized to evaluate competitive architectures in a Family Computer Study by the U.S. Army and Navy.

(6) SMITE which is based on TRW's development work (PRES79, TRW79) for USAF/RADC. SMITE is a high level computer description language that is based on many of the modern programming practices for code development outlined in the IBM Structured Programming Manuals (NAUG75).

Several systems have been developed using microprogrammable computers and hardware description language. Some examples of these systems are described below.

2.3.4 Systems Utilizing Microprogrammable Computers

Since the early 70's microprogrammable computers have been described in the literature as tools for emulating computer systems.

Rosin et al (ROSI72) has described the benefits of microprogramming and emulation utilizing microprogrammable computers. This early work has led to the establishment of a number of environments on microprogrammable computers.

Diebolt (DIEB) utilized the general purpose microprogrammable processor (QM-1) to emulate a third generation Univac Series 90 processor. This study was undertaken in order to demonstrate that computer architectures can be emulated under a common microprogram architecture and to identify features and functions which directly contribute to performance and micro-primitives common to target emulators.

Gallenson (GALL78) describes the Programming Research Instrument (PRIM) that utilizes an MLP-900 microprocessor to provide a working environment in which a user can implement a computer in micro-code and run that computer in a chosen program environment. The PRIM system allows interactive execution and debugging of code for emulated computer systems by a number of users. The design of the PRIM environment recognizes the need to isolate the microprogrammable computer from other resources in the system.

Beach (BEAC) reports on the Microprogrammable Multi-Processor System (MMP) which utilizes a series of CDC 56 processors to perform system emulations on a planned target system to investigate the interoperability characteristics of existing or planned target systems. Demco and Marsland (DEMC75) utilized a Nanodata QM-1 to develop a complete emulation of a PDP-11 instruction set and developed a monitor which supports multiple concurrent emulation on the QM-1 machine. This monitor is supported in control store of the microprogrammable computer.

It provides functions similar to an operating system such as interrupt handling, I/O support to member emulators, and supervisory, debug, and control capabilities to an operator.

Burkhardt (BURK) describes the development of an operation system written in Pascal which will facilitate the development of software for the QM-1 microprogrammable computer.

In each of these studies the merits of utilizing dynamically microprogrammable computers has been established. These systems concentrate on using microprogrammable computers to emulate existing computer systems.

The development of data base machine architectures can utilize some of the capabilities from these systems. These capabilities need to be integrated with other tools and procedures that can help develop a data base machine architecture. For instance, one way to test an architecture, following its description in a hardware description language, is by extensive verification procedures. A microprogrammable computer can be used to execute the description of the data base machine architecture. The user has control over each state of execution as it executes at the micro-instruction level. A specialized verification technique has been proposed by Crocker (CROC77) to examine states of execution of a machine during execution on a microprogrammable computer. This technique can help provide the ability to verify the function provided by data base machine architectures.

One way to measure the effectiveness of a data base machine architecture is to determine how it affects such factors as reliability, maintainability, testability, etc. for the DBMS software module it is replacing. A framework for measuring software quality factors is

included in work completed by McCall and Cavano (MCCA79,CAVA79). A set of metrics has been established for eleven different quality factors. These same metrics, if applied to the description of the new DBMS hardware architecture, may be used to judge how the new architecture affects a specific factor. Such an idea is now being explored by Vermuri et al (VERM80).

The development of new data base machine architecture can also benefit from current research in virtual machine technology. The advantages of using this technology to examine new approaches for developing these architectures is described next.

2.4 VIRTUAL MACHINE TECHNOLOGY

The concept of a virtual machine was described by Goldberg (GOLD73). He defined a virtual machine as a "hardware-software duplicate of a real existing computer system in which a statistically dominent subset of the virtual processors instructions execute directly on the host computer in native mode."

Illustrated in Figure 2-5 is the concept of a virtual machine monitor (VMM). The VMM extends the bare machine into a number of virtual machines, each capable of supporting the same instruction sets as the bare machine. Each virtual machine can now support the same set of software that executes on the bare machine. The resources for all virtual machines are maintained by the VMM. These resources are identified as a virtual machine is created. However, only the VMM can allocate the resources.

On the virtual machine, the set of instructions may be partitioned in two subsets: nonpriviledged which are executed directly on the host

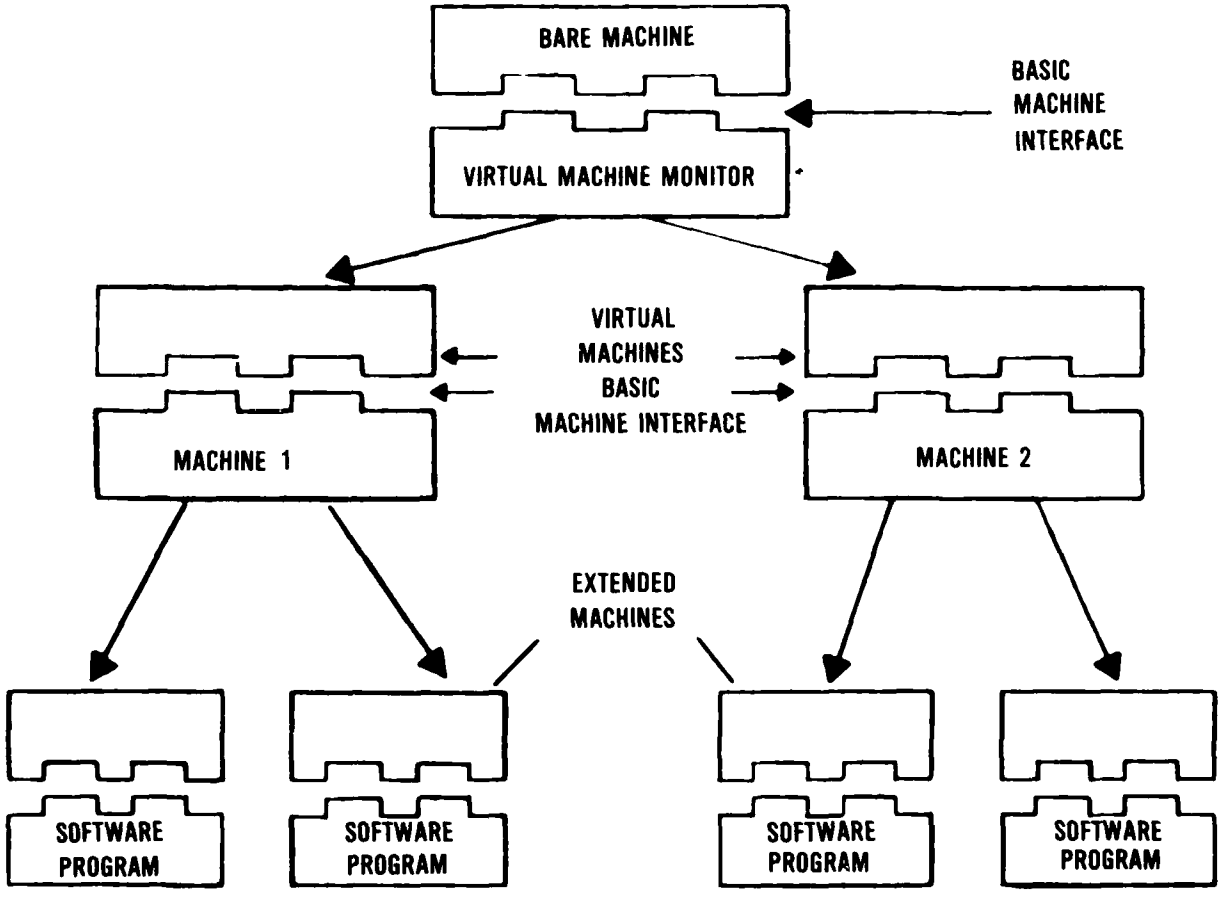


ILLUSTRATION OF A VIRTUAL MACHINE MONITOR

FIGURE 2-5

processor in native mode, and the privileged operations which make gross changes to the state of the machine. These instructions can include requests for the allocations of resources for I/O operations, those requesting changes to mode of the machine, or those requesting changes to a machines storage protection. If the virtual machine attempts to execute a privileged instruction, it is trapped and control is passed to the VMM. The monitor then simulates the instruction in a manner consistent with the overall architecture of the virtual machine. The actual request is then issued by the VMM to the real resource. In this manner, the resource allocation function becomes the responsibility of the VMM.

A programming environment consisting of virtual machines provides several advantages including:

(1) the ability of the system programmer to make changes to system software without affecting other machines,

(2) improved system reliability since one machine may fault without affecting the operation of the other machines,

(3) an environment that exhibits extraordinary flexibility by permitting incompatible software systems to be tied together compatibly as demonstrated by Donovan (DON075),

(4) a means for software development using a dynamically microprogrammable computer as a host machine as described by Flink (FLIN77), and

(5) the capability to model a network of processors as cited by Goldberg (GOLD74) where messages between virtual processors may be routed over simulated transmission control units or they may be sent out over physical communication lines before getting to the receiving virtual machine.

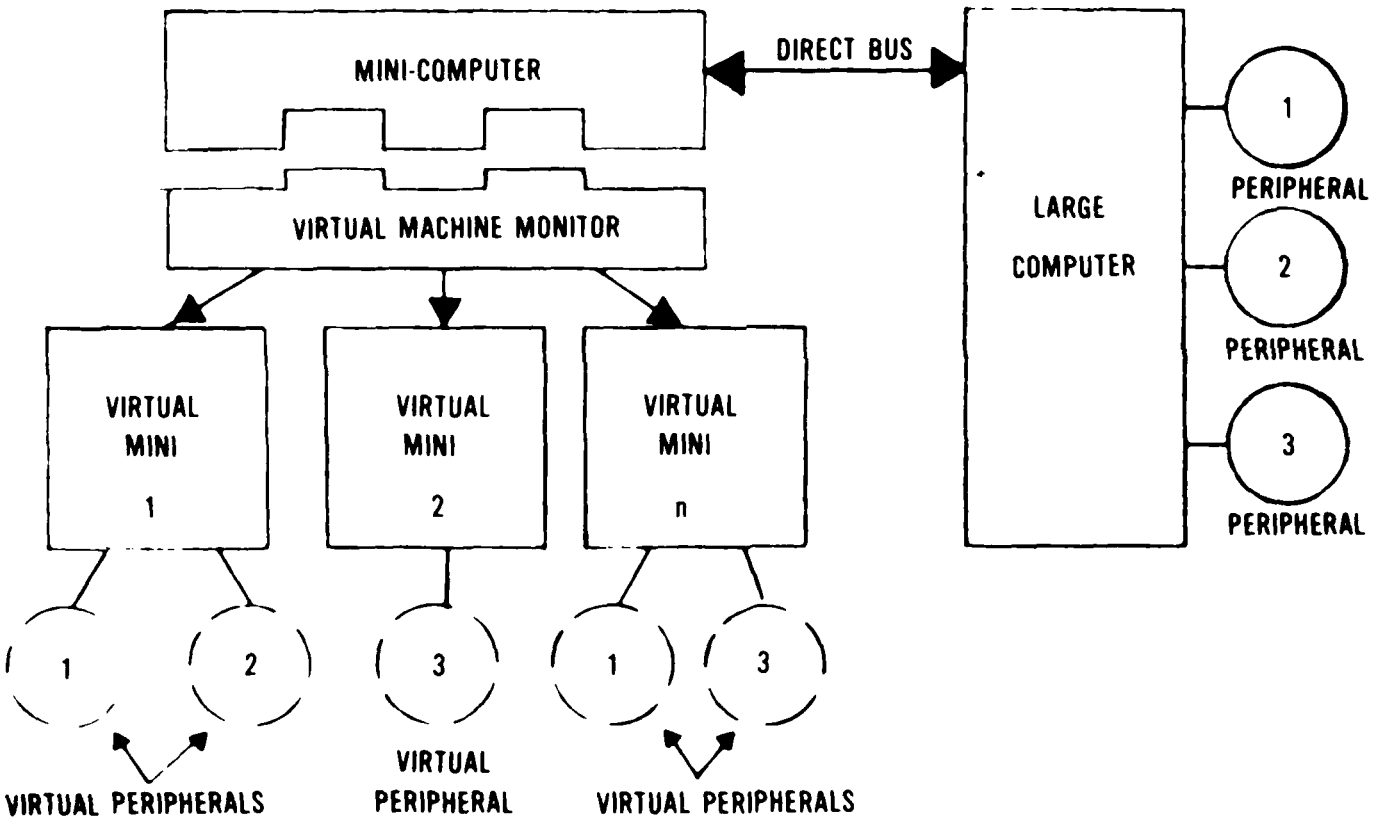


ILLUSTRATION OF VIRTUAL PERIPHERALS

FIGURE 2-6

The advantages of a programming environment using virtual machines have been explored in the data base management area by Donovan and Madnick (DON077). They have developed the Generalized Management Information System (GMIS) that provides a data base management capability on a single machine architecture. This system uses virtual machine techniques as an integrating mechanism to provide:

(1) a mechanism for multi-user coordination of access and update to a central data base,

(2) a mechanism for creating an environment where several different modeling facilities can access the same data base, and

(3) a mechanism for creating an environment where several different and potentially incompatible data management systems can all be accessed by the same user models or facilities.

The goal of the GMIS system is to improve performance of a number of data base management activities in a single computer environment. This work was further extended (MADN78) to demonstrate the communication ability among clusters of virtual machines.

Virtual machines can also be utilized to access virtual peripherals as cited in the work by Gagliardi and Jain (GAGL77). Their systems interface a mini-computer (PDP 11/40) to a large computer (PDP-20) via a direct bus connection.

This work is significant because it provides a capability to isolate a machine from resources needed for I/O operations. This is precisely a requirement needed to isolate the execution of data base machine architectures on a micro-programmable computer from external

resources needed during their execution.

The advantages provided by virtual machines can help provide many of the capabilities needed in a facility that can help develop data base machine architecture.

2.5 SUMMARY

The role of data base management and architecture has taken on greater meaning since the development of inexpensive new hardware technologies. Generalized microprogrammable computers have been developed that permit a wide variety of architectures to be emulated. Work in the area of virtual machines make it feasible to integrate various tools, procedures, and resources in common environments. Each of these advances has contributed a set of specifications for a facility that can help develop data base machine architectures. The problems of developing data base machine architectures that can be helped with this facility and the research methodology used for its specification are described in the next chapter.

CHAPTER III

PROBLEM DEFINITION AND RESEARCH METHODOLOGY

3.1 INTRODUCTION

Described in this chapter are the problems concerned with choosing which data base management functions, now performed in software on sequential computers, should be moved to new hardware architectures. These problems have been examined, and a research methodology has been developed to help solve them. Part of this methodology includes defining a facility that can assist in developing special purpose DBMS function architectures. Included in this proposed facility are the set of specialized tools and components needed to help a user develop these SPFA's.

3.2 PROBLEM DEFINITION

The lower costs for computer hardware have made it feasible to initiate efforts to design new architectures for data base management functions. These new architectures may, in the future, be combined into specialized machines that provide the complete data base management function for an organization. However, choosing which data base management functions should be moved into hardware is not an easy task. In order to help make this choice, several questions must be examined:

(1) what are the important factors to consider when choosing a function,

(2) what architecture is most nearly optimal for a specific DBMS function, and

(3) how will new hardware technology affect the choice of the architecture in terms of performance, cost, reliability, etc?

Several types of DBMS functions are candidates to be moved into hardware. An examination of a data base management system indicates that it is composed of several types of functions. First, a set of basic functions are provided by a DBMS which perform commands directed by an application program. These functions are used to manipulate data into a form acceptable to the application program. A second set of functions maintain data in a data dictionary or a data base. The functions permit a logical expression of the data base and maintain a physical access to the stored data. Next, a set of functions provides a set of user interface capabilities. Typically, these functions are provided by query generation modules or request generation modules. Finally, a set of application modules are used to support functions. These modules provide various editing types of capabilities for a DBMS. Classes of these functions include sort or merge programs.

Each of these various types of functions are used to support the data base management activity of an organization. However, as organizations seek approaches to upgrade their DBMS's, by introducing new DBM architectures, it is not clear which of these functions should be moved to hardware. In addition, in making this choice, an organization will want to seek the optimal DBMS architecture that can provide an improved DBMS function and still provide compatibility with their current software DBMS application programs. If not, the large cost investment in these programs may suggest that a new data base machine architecture is not feasible. Thus, a procedure is needed that can assist in choosing

candidate DBMS functions, design candidate hardware architectures for these functions, and provide a capability to determine which will be favorably integrated into current DBMS's.

For each data base management function that is a candidate to be moved to hardware, several architectural options may be available. In order to evaluate each option, the actual hardware may have to be built. If more than one architecture is being considered, several options may have to be built. Once these hardware options are built, procedures to test and evaluate them need to be established. However, if many DBM architecture options exist, the actual hardware construction may not always be feasible because of:

(1) the expense of actually developing a number of hardware options,

(2) time constraints, and

(3) inability to easily alter each DBM architecture option after it has been built.

Thus, procedures are needed that assist in choosing optimal DBM architectures prior to actually having them built.

Finally, the actual hardware technology used to perform operations needed for a DBM architecture will affect its performance, cost and reliability. The technology chosen by a user depends on specific user application requirements. However, a user needs to know the effects of a specific technology choice. For instance, one technology may sacrifice speed for more reliability for the DBM architecture. Another may improve performance but cost more. In order to help evaluate a DBM architecture option, a procedure is needed that can

examine various strategies for a DBM architecture performance and choose the optimal strategy for a specific application.

Each of the above problems has been examined in this research. The result has been to specify a facility with a specialized set of tools and components, designed specifically to help users develop data base machine architectures as emulated machines. This proposed facility permits a user to examine a data base management function, examine new hardware architectures for this function, and tailor the development of a specific architecture to personal needs. A methodology for utilizing this facility is also proposed. Included in this methodology are several processes.

The approach used to specify this facility and to demonstrate its applicability for developing data base machine architectures is outlined *in section 3.3.*

3.3 RESEARCH METHODOLOGY

Part of the research leading to the specification of a facility to help develop data base machine architectures has led to an examination of what constitutes a data base machine. Liuzzi and Berra (LIUZ78) have defined special purpose function architectures (SPFA's) as subsets of the architecture of a data base machine. The research methodology outlined in Figure 3-1 includes specifying a Data Base Machine Architecture Development (DMAD) Facility that can be used to develop these special purpose function architectures.

The specification of the DMAD Facility consists of:

(1) designing the architectural interfaces among a set of hardware components that comprise the facility,

(2) identifying a set of unique tools, in the facility, used to develop special purpose function architectures, and

(3) identifying a methodology to utilize the facility.

This is followed by a demonstration of a specific implementation of the DMAD Facility. This demonstration includes the actual development of two different special purpose function architectures that perform the same DBMS function. It is shown how the facility is utilized to develop and evaluate each approach.

The organization of this research methodology leads to an overall research contribution that combines several current research disciplines into defining a set of tools/components used to develop DBMS special purpose function architectures. This research methodology is now described in more detail below.

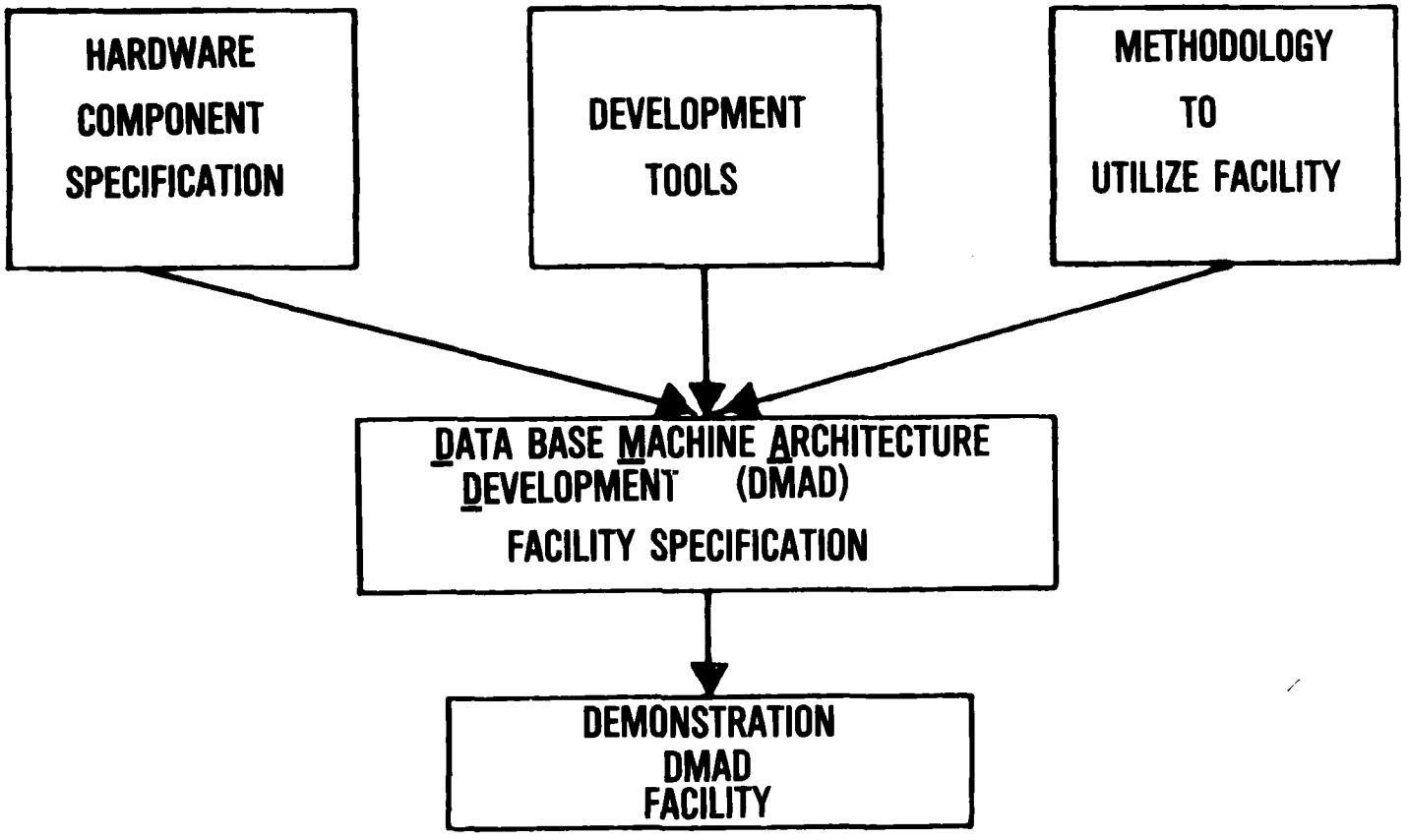
3.3.1 Hardware Components

A set of hardware components have been identified for the DMAD Facility. These components consist of several machines that perform specific DMAD Facility functions. These components include:

(1) a machine that is capable of executing, as an emulation, the architectural description of a SPFA,

(2) a machine that can provide a programming environment, along with a set of resources to support execution of machines in the facility,

(3) a machine that provides a set of library services for requests that utilize the DMAD Facility. This machine is used to identify a set of facility configuration requirements for input requests, and



RESEARCH METHODOLOGY

FIGURE 3-1

(4) a machine that both manages and configures the DMAD Facility for each request.

3.3.2 Development Tool Specification

Several tools have been identified for the DMAD Facility to help develop a special purpose function architecture. These tools consist of:

(1) a DMAD process description language,

(2) a DMAD hardware description language,

(3) a DMAD execution control module,

(4) a data base machine architecture configuration array,

(5) a realization timing control array, and

(6) a virtual data base machine monitor.

The process description language identifies the type of SPFA development process requested within the facility. The hardware description language defines, in a high level language, the architectural constructs of the SPFA. A DMAD execution control module is used to control execution and provide specialized tools for each machine in execution. The data base machine architecture configuration array identifies all machines and their configuration requirements needed to execute in the DMAD Facility. The realization timing control array is used to assign timings for various operations of each SPFA during execution. Finally, the virtual data base machine monitor is a tool that provides the means to substitute a SPFA for the DBMS function it performs. These tools are available to the DMAD Facility user as he/she develops a SPFA.

3.3.3 Methodology to Utilize the DMAD Facility

The methodology to utilize the DMAD Facility is introduced by a process flow model which identifies a set of processes that can be used to develop a SPFA. Each process in this model requires a set of functions to complete. These functions are performed by a series of procedures using the tools/components available in the facility.

3.3.4 Demonstration of a Specific Implementation of a DMAD Facility

A specific implementation of a DMAD Facility consists of a set of tools/components that include a QM-1 microprogrammable computer and the SMITE hardware description language. The demonstration of the facility illustrates how it can be used to develop special purpose function architectures. This demonstration includes:

(1) illustration of how the facility chooses candidate DBMS functions,

(2) development of a "merge" SPFA based on Hollaar's (HOLL75) "merge" processor,

(3) development of a "merge" SPFA that is based on Stellhorn's inverted file processor and uses Batcher's odd-even merge network,

(4) execution of both SPFA's as "merge" machines,

(5) actual use of the facility to produce various types of comparative evaluation data for the merge machines, and

(6) demonstration of use of the facility to compare alternative approaches for performing the merge DBMS function in hardware, firmware and software, in a common environment.

3.4 SUMMARY

Described in this chapter are the problems concerned with developing data base machine architectures for specific applications. A research methodology is then outlined that addresses these problems and introduces the concept of a DMAD Facility. Described in the next chapter is the basic description of functional DMAD components and how they can be used to develop a SPFA. Described in CHAPTER V is a detailed examination of how the facility components, tools and procedures could be used to develop SPFA's. Described in CHAPTERS VI and VII is a demonstration of the feasibility of a DMAD Facility. A specific instance of the facility is used to develop several SPFA's concerned with the merge function. Finally, described in CHAPTER VIII are results and conclusions of this research and several suggestions on where future research sould be initiated.

CHAPTER IV

INTRODUCTION OF A DATA BASE MACHINE ARCHITECTURE DEVELOPMENT FACILITY

4.1 INTRODUCTION

Described in this chapter is the design of a Data Base Machine Architecture Development (DMAD) Facility. This facility can be used to:

(1) help determine what kinds of DBMS functions are candidates to be moved from software to hardware,

(2) help develop DBMS special purpose function architectures (SPFA's),

(3) help preserve DBMS software application program investment by introducing the SPFA's in a selective integration (evolutionary approach) to existing data base management systems,

(4) help incorporate cost effective, new hardware technology into data base management, and

(5) assist in the specification of future Data Base Machines.

A methodology to utilize this facility is also described in this chapter. This methodology describes a set of basic processes that use the tools/components of the DMAD Facility to develop SPFA's.

This methodology includes:

(1) a detailed specification of the functional flow for each process within the DMAD facility,

(2) identification of a set of functions for each process, and

(3) procedures to accomplish these functions using specialized DMAD facility components/tools.

The remainder of this chapter describes this methodology and the set of architectural/interfaces that comprise the facility.

4.2 METHODOLOGY TO DEVELOP A SPFA USING DMAD FACILITY

A methodology to develop a special purpose function architecture requires the following set of processes which are illustrated in Figure 4-1:

(1) Select candidate function,

(2) SPFA create,

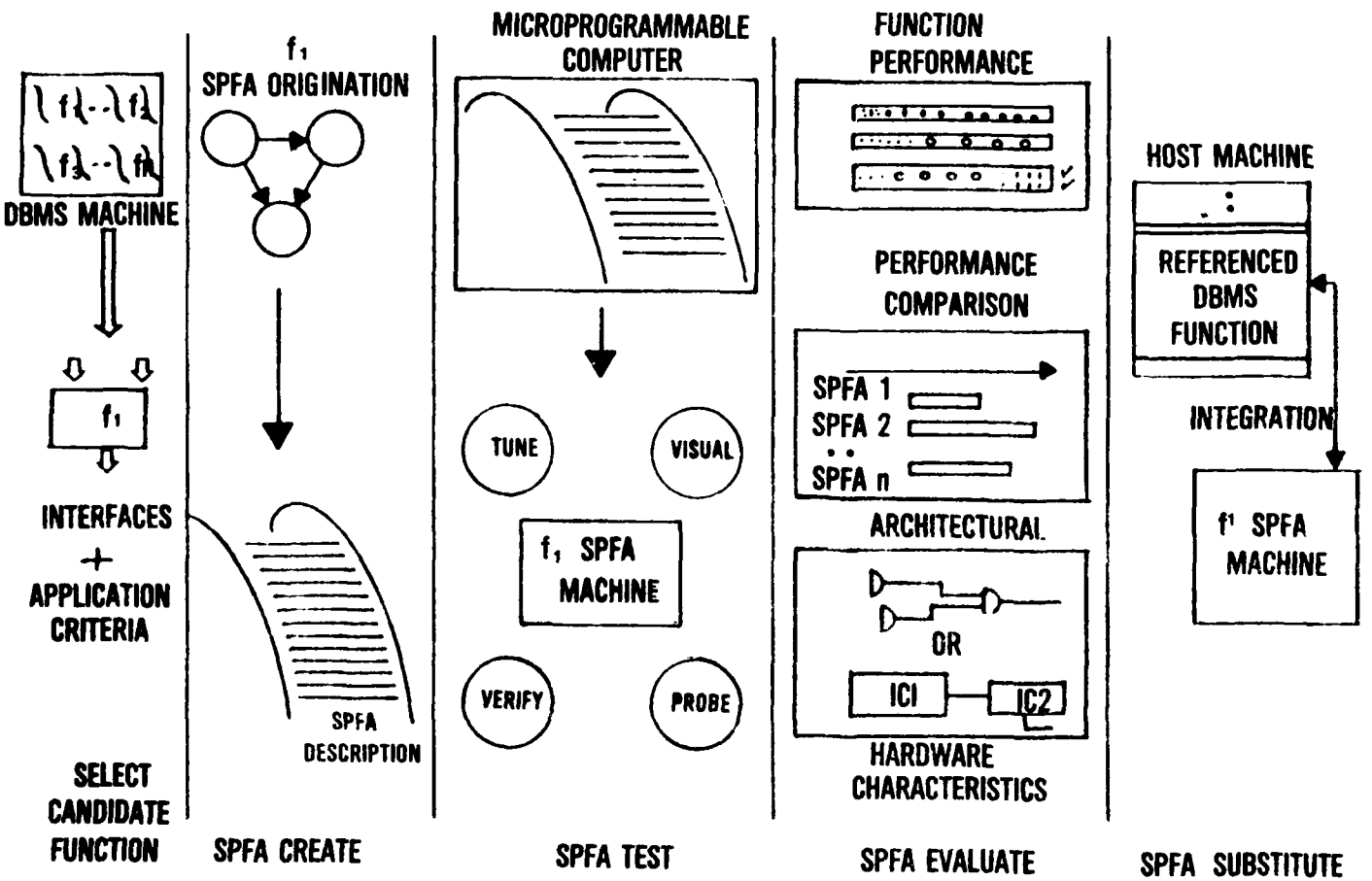
(3) SPFA test,

(4) SPFA evaluation, and

(5) SPFA substitute

The select candidate function process utilizes the facility to help determine those DBMS software functions that are candidates for replacement as hardware architecture SPFA's. Once a candidate function is selected, the facility is used to develop SPFA's which can perform the DBMS function.

The creation process transforms a description of each SPFA from a set of architectural considerations to a set of language statements.



DEVELOPMENT PROCESSES

FIGURE 4-1

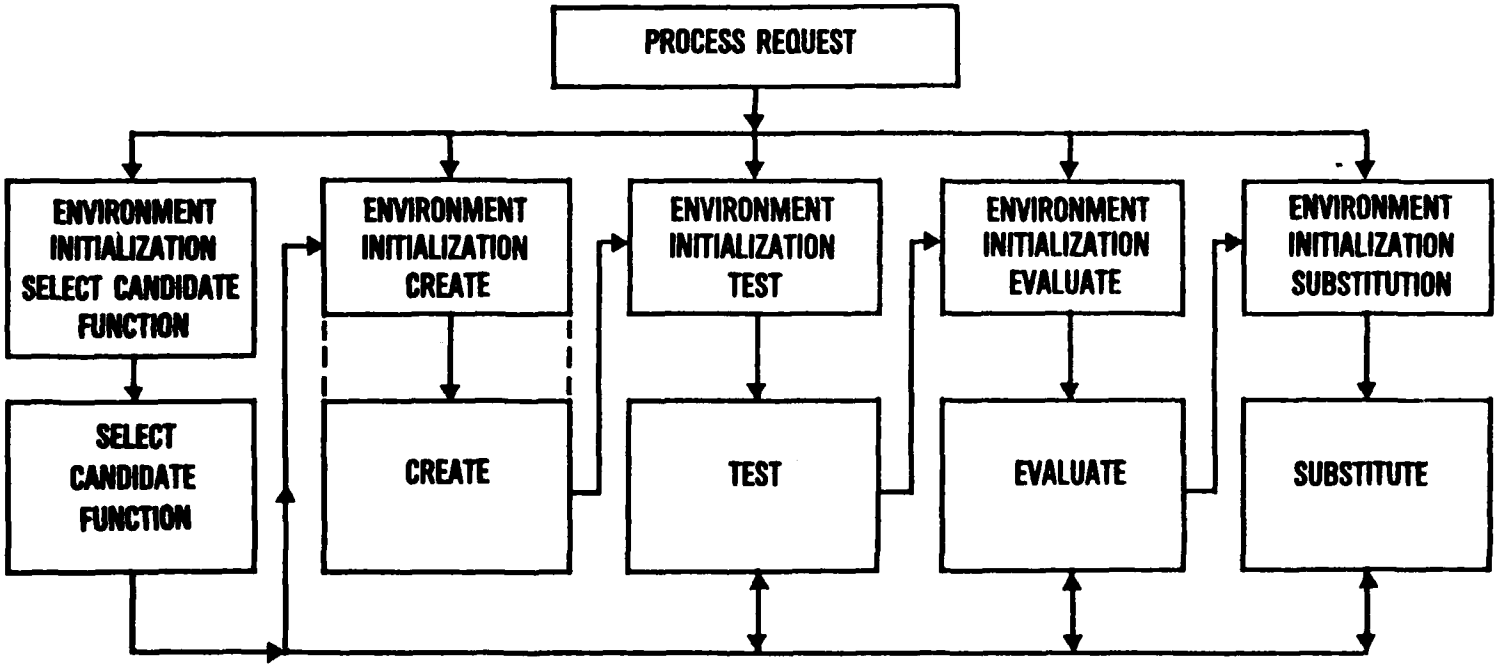
This set of language statements represents a functional description of an architecture that performs the DBMS function.

The test process functionally verifys that the SPFA performs the desired DBMS function. This process permits the data base machine architect to examine the architectural constructs of the SPFA to insure it meets its design goals.

The evaluate process enables the data base machine architect to evaluate competing SPFA's. An architecture evaluation is conducted by using the facility to generate performance timings of the SPFA's. These timings are based on utilizing an assumed set of hardware characteristics to perform operations required by each SPFA.

Finally, the substitution process is composed of a set of procedures to enable the data base machine architect to selectively integrate a SPFA within a data base management system capability. The substitution process helps the data base machine architect assess the system impact of having a DBMS software function replaced by hardware.

The complete development using the facility is illustrated by a process flow model in Figure 4-2. The flow of this model indicates the DMAD Facility is initialized for each process request. This initialization configures a machine and tools for each process. Several feedback loops are provided in this model to allow refinements during the development of the SPFA. These loops permit reuse of the complete set of DMAD Facility tools for all processes. For instance, after an evaluation process is completed, the data base machine architect may choose to alter a SPFA by modifying a portion of the architecture



DEVELOPMENT PROCESS FLOW MODEL

FIGURE 4-2

description. This may result in the recreation of the SPFA. Similarly, a single process can be repeated interactively so an exhaustive series of tests or evaluations can be performed.

In addition, the final process, substitution, permits the data base machine architect to assess the impact on the DBMS system of the newly developed SPFA. This process is used to integrate the SPFA within the DBMS and utilize the facility to help determine potential problems. The data collected following this process can help determine if the function is a logical candidate to be moved to hardware for a specific application.

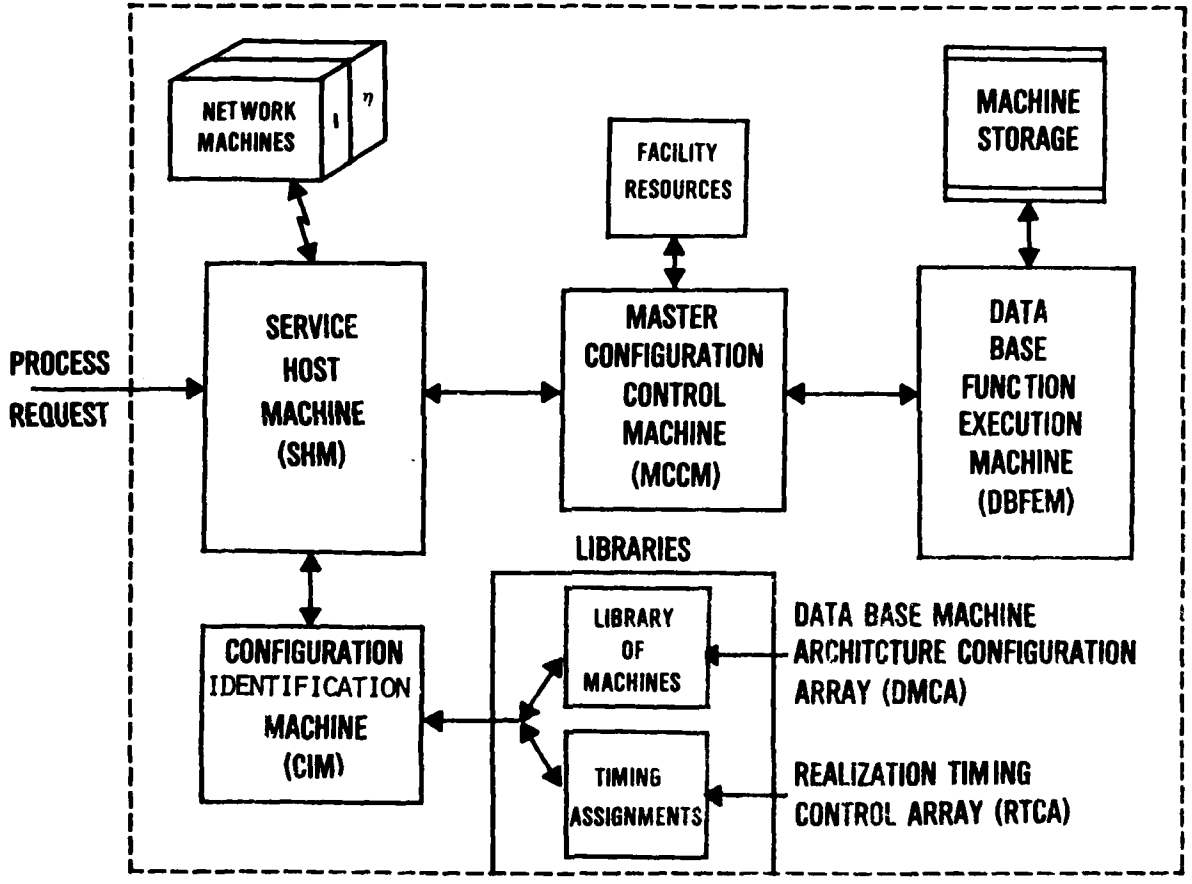
The use of the process flow model also permits a user to tailor his/her development to a specific application. This helps insure that the SPFA meets the unique requirements of the application.

4.3 ARCHITECTURAL/INTERFACES OF THE DMAD FACILITY

The architectural/interfaces of a Data Base Machine Architecture Development Facility (DMAD) are illustrated in Figure 4-3. The DMAD Facility consists of the following components:

(1) a Service Host Machine (SHM) that is responsible for monitoring requests in the DMAD Facility, staging input for the Data Base Function Execution Machine (DBFEM), and providing a programming environment to described SPFA's. The SHM interfaces to network machines which may help execution of a SPFA.

(2) a Data Base Function Execution Machine (DBFEM) that is responsible for hosting the execution of machines in the facility. This machine serves as a back-end to the SHM. This machine is capable of emulating a variety of computer architectures.



ARCHITECTURAL/INTERFACES OF THE DMAD FACILITY

FIGURE 4-3

(3) a Master Configuration Control Machine (MCCM) that interfaces the SHM and DBFEM. This machine acts as the configuration manager for process requests to the DMAD Facility. In this capacity, this machine controls resources needed to support execution of a machine on the DBFEM,

(4) a Configuration Identification Machine (CIM) that is used to identify the DMAD Facility configuration requirements for each process request. The CIM interfaces to the SHM and mantains two libraries. The first defines all machines that can execute within the DMAD facility and the second maintains timings for operations performed by the SPFA machines.

Within this facility, the DBFEM is to be used exclusively as a tool to emulate various machines. The supporting components in the facility are used to assist each emulation with a set of specialized tools to help perform each of the development processes.

4.4 ILLUSTRATION OF UTILIZATION OF DMAD FACILITY

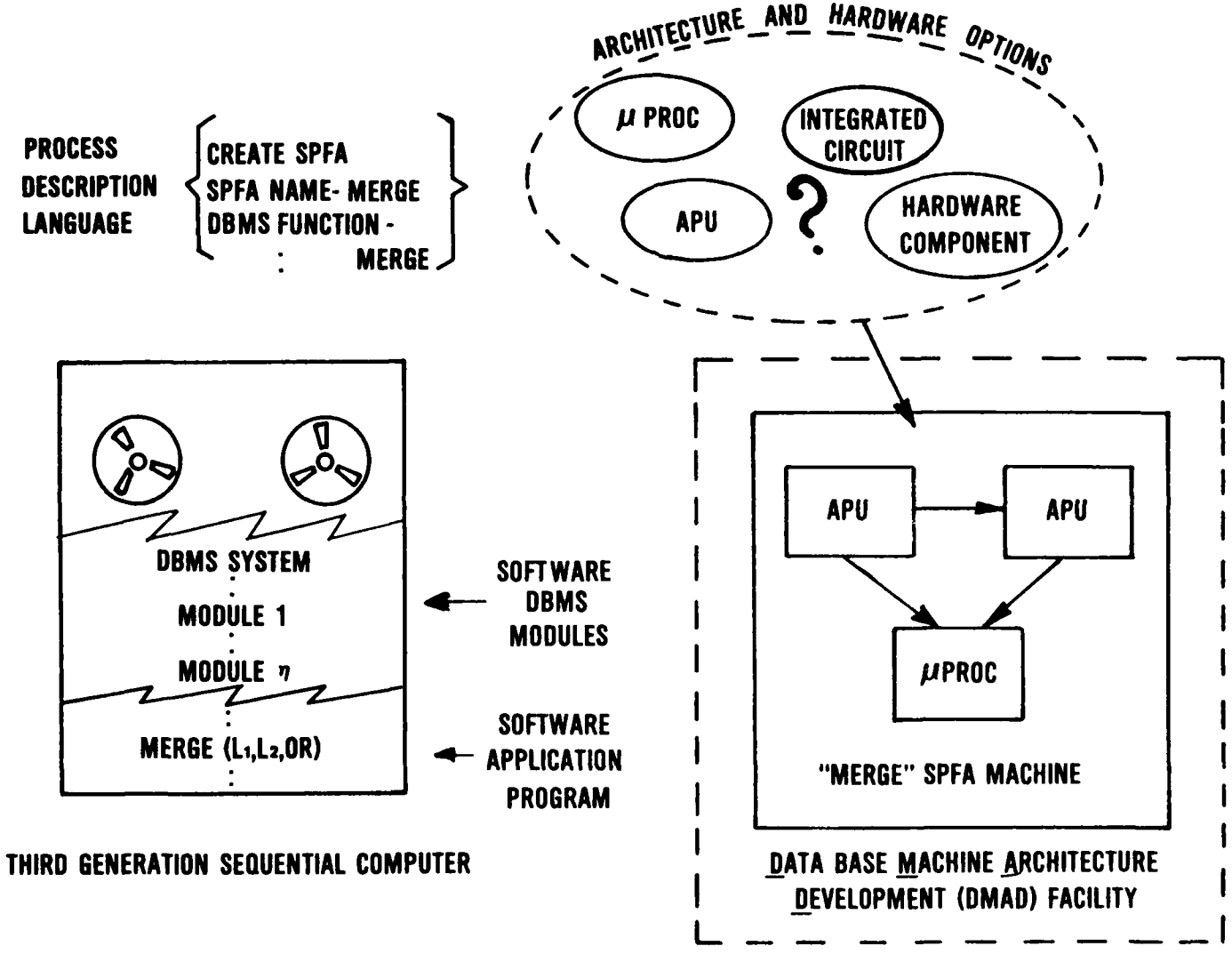
A typical utilization of the DMAD Facility is shown in Figure 4-4. An organization requires improving the performance and reliability for merging lists for its present applications. This "MERGE" function is currently being performed in software, as part of a DBMS, on a sequential computer. However, several competitive new merge hardware architectures can also perform this function. This organization needs to choose the merge hardware architecture that can optimize the overall performance and reliability of this application.

Using a DMAD Facility the merge hardware architectures are created as SPFA machines to perform the DBMS "MERGE" function. For instance, illustrated in Figure 4-4 is a "MERGE" SPFA machine that is created from among several architecture options and is introduced to the DMAD Facility. The SPFA is described by a hardware description language. This description is compiled and debugged to produce an executable version. When the architect is satisfied with the description, it is configured in the DMAD Facility to produce the SPFA machine. During execution, the DBMA has complete control of the SPFA machine which permits him/her to examine all states of execution. Testing in this environment is done with a set of tools to verify that the SPFA performs its intended DBMS function.

After testing, an evaluation of the SPFA is performed. This evaluation consists of accumulating the time needed by the SPFA machine for a sequence of its operations. The timing of these operations is chosen from several hardware characteristics included as possible implementations of SPFA.

For instance, in this example, since both performance and reliability improvements are sought, the DBMA can examine the performance of hardware technologies that are highly reliable for the "MERGE" machine's operations. In order to assess the effect of varying these choices, one or more realization assumptions can be described. Each choice becomes a separate realization of the SPFA and are used to generate trade off data on performance and reliability.

Once performance and reliability data are established for a specific SPFA description, the same procedure described above can be repeated by varying some architectural features of the original

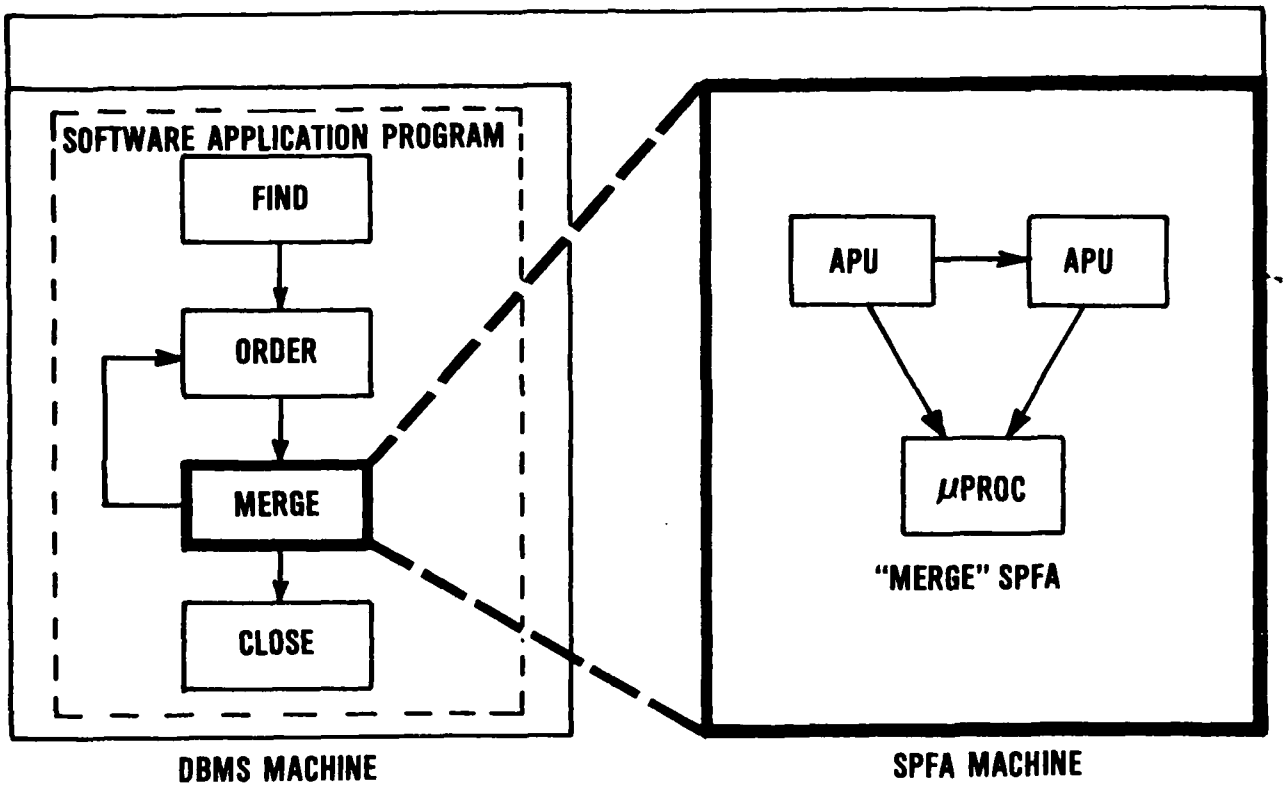


DEVELOPING A SPECIAL PURPOSE FUNCTION ARCHITECTURE

FIGURE 4-4

SPFA description or developing one of the competitive "MERGE" SPFA's. This process can continue for a number of architectural options that may be available for this function.

Once a "MERGE" SPFA is chosen, the next development stage can be its substitution within a current DBMS. This process can be performed utilizing a DBFEM, as illustrated in Figure 4-5. First, a computer system that can support a set of data base management functions is referred to as a DBMS machine. Next, assume this DBMS machine is emulated to execute on the DBFEM. One function available on the DBMS machine is the "MERGE" function. A DBMS software application program (SAP) is chosen to execute on the DBMS machine and call the services of the DBMS. The SAP typically calls a sequence of DBMS functions such as FIND, ORDER, MERGE, CLOSE, etc., as illustrated in Figure 4-5. Whenever the "MERGE" function is called, the option chosen for the "MERGE" SPFA machine is executed instead of the original "MERGE" function. The data base machine architect can now examine the impact of the selective integration of the SPFA. This permits assessing the hardware/software tradeoff issues pertinent to this integration. A positive assessment of the SPFA's integration may lead the organization to choose to actually build a prototype SPFA from its description. The results gathered in all development processes can now be made available to the hardware developer to assist in the actual hardware prototyping.



DATA BASE FUNCTION EXECUTION MACHINE

SUBSTITUTION OF A SPECIAL PURPOSE FUNCTION ARCHITECTURE

FIGURE 4-5

4.5 SPFA DEVELOPMENT PROCESS FUNCTIONS

Described in this section are a set of functions that utilize the DMAD Facility to help choose a DBMS function and develop a SPFA. The development is separated into:

(1) identification of candidate DBMS functions,

(2) origination of a SPFA, and

(3) execution of a SPFA.

4.5.1 Identifying Candidate DBMS Functions

The identification of candidate DBMS functions is made by examining typical requirements of a given application. These requirements can include usage of the function, ability to provide a well defined function, and a need to improve performance and costs. Candidate functions are examined using the select candidate function process.

4.5.1.1 Select Candidate Function Process

This process utilizes the DMAD Facility to select the DBMS functions that are candidates to be moved from software to hardware. This process utilizes the facility to help a data base machine architect make the choice of potential DBMS candidates that are logical choices and would benefit from being moved from software to hardware. This process requires the following functions:

(1) DBMS machine identification,

(2) DBMS machine execution, and

(3) DBMS module data collection.

The DBMS machine identification function is used to enter emulations of current machines that support data base management systems

in the DMAD Facility. These machines are entered into the DMCA. Each entry assigns the resources that permit these machines to be configured for execution in the facility.

The actual execution of a DBMS machine in the facility can be made following configuration. Each DBMS machine is configured with system support software, a DBMS, and application programs. A set of DBMS functions are then identified as functions required by the application programs. These functions, composed of one or more software modules, are candiates to be replaced by hardware.

A DBMS module collection function utilizes the facility to collect statistics on usage of each of the DBMS functions. These data are collected using the facility by either examining the state of the DBMS machine each time a function is called or using software data collection monitors to help compute utilization data during the execution of the DBMS machine. Once a function is identified the facility is used to define its interfaces within the DBMS system. These interfaces include the inputs and output required by the function and its complexity within the DBMS.

In addition to utilization statistics, the facility can also be used to examine other factors that may help choose a function. These are factors about each software module that affect the quality of the DBMS such as reliability, maintainability, etc. Metrics for these factors can be computed using the results of a study by McCall (MCCA79) regarding software quality factors. The set of data collected is then examined to determine potential candidates to be moved into hardware as SPFA's.

4.5.2 Origination of a SPFA

The origination of a SPFA represents the specification of the design of a special purpose function architecture that performs the DBMS function

in hardware. This specification is then translated into a language that describes the architectural constructs, flow of information and operation interdependencies of the SPFA. A SPFA can be originated by issuing a create process request to the DMAD Facility.

4.5.2.1 Create Process Functions

The objective of a create process is to translate a hardware architectural description of a SPFA into an executable SPFA machine. This process consists of two functions:

(1) SPFA description development, and

(2) SPFA introduction.

The SPFA description development function requires the specification of a set of architectural constructs, that when executed as a machine, perform a DBMS function. These constructs can be specified using a hardware description language that defines a machine representation of a SPFA.

The SPFA introduction function identifies a SPFA to the DMAD Facility. This function identifies the set of resources needed to execute the SPFA in the facility. This function is performed by introducing a new entry in the DMCA for a SPFA. Once this entry is made in the DMCA a SPFA is available for execution.

4.5.3 Execution of a SPFA

The execution of a SPFA represents the ability to transform the DMAD Facility into a SPFA machine. The transformation of the DMAD Facility requires defining the set of configuration resources needed to support execution of a SPFA for a process request.

The notion of this transformation is illustrated in Figure 4-6 where the DMAD facility is transformed into one of several SPFA machines. For instance, the facility is transformed into a "FIND" machine, a "MERGE" machine, or a "Data Dictionary" machine.

This transformation begins with a machine identification procedure. This procedure chooses the machine description from a set of other machine descriptions in the DMCA. This array contains all available machines, their machine loadable location and their special resource execution requirements. Once the machine has been identified, a set of configuration resources are defined. These resources are used to configure the DMAD facility for execution. Once the facility has been transformed, the testing, evaluation, and/or substitution processes can be performed.

4.5.3.1 Testing Process Functions

The objective of the testing process is to determine if the SPFA machine accurately performs the desired DBMS function. The testing process consists of these functions:

(1) execution of the SPFA machine using test cases,

(2) verification of proper sequence SPFA machine states, and

(3) debugging the SPFA machine.

The execution of the SPFA machine insures that the loadable version of the SPFA machine executes correctly. In order to begin execution, a set of test cases can be specified. These test cases consist of specific input to the SPFA machine to insure it executes properly.

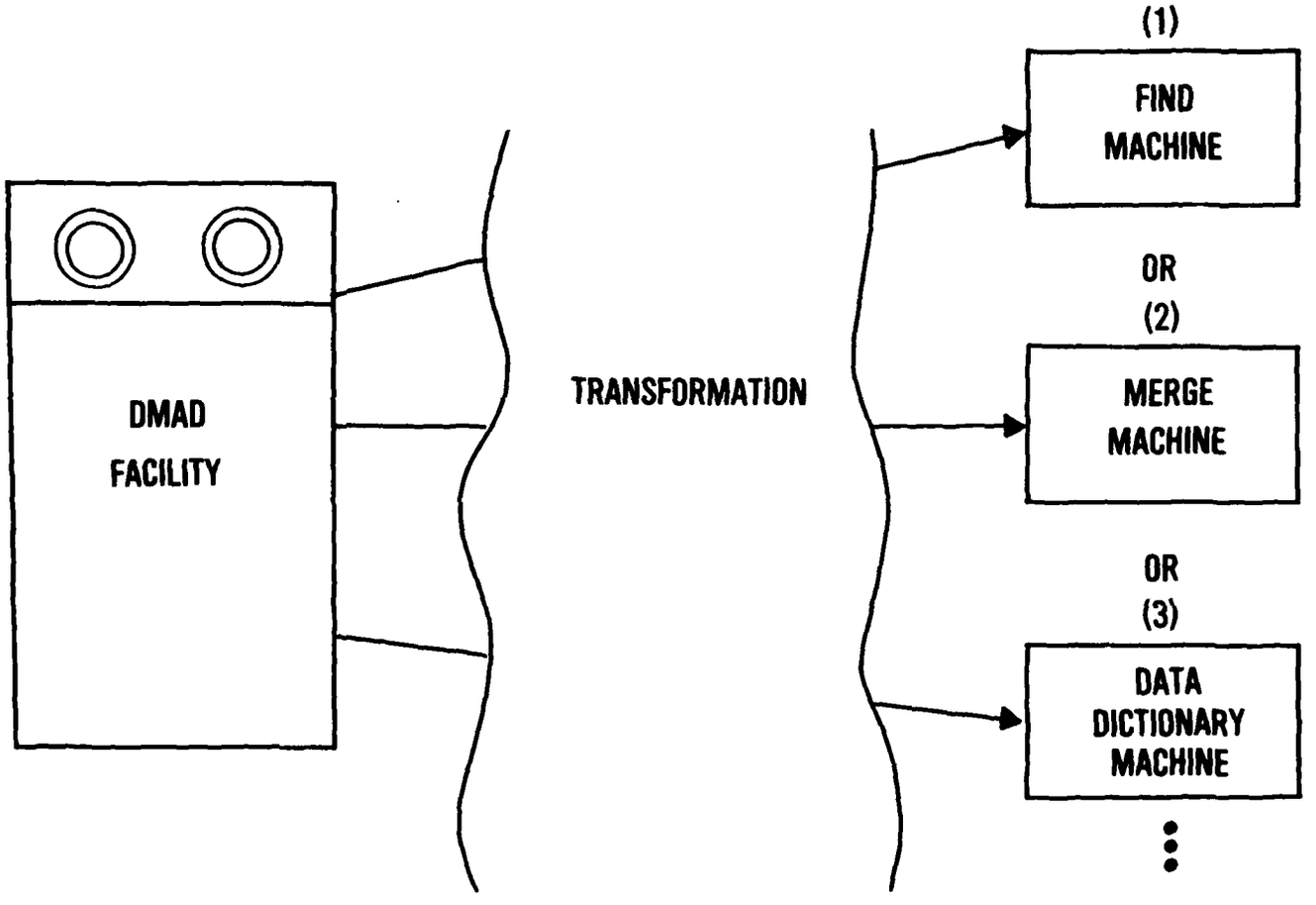


ILLUSTRATION OF TRANSFORMATION OF DMAD FACILITY

FIGURE 4-6

The SPFA machine executes by moving from state to state. A state can be specified at the SPFA source description level or at the micro-instruction level. The micro-instruction level permits identification of states at a much lower level than the source level. The level may be needed for detailed verification or debugging the SPFA machine.

One verification technique for SPFA machines is made by examining the state of the machine at a given instant. This examination can be conducted by establishing a breakpoint in the SPFA description. When this breakpoint is reached during execution of the SPFA machine, the data base machine architect performs an extensive verification of the state of the SPFA machine. The SPFA registers, information resources, and control indicators are examined. If they conform to pre-selected values, the state verification of the SPFA machine is established. However, if an error or inconsistency is found at one of these breakpoints, then the SPFA machine may not be verified and debugging procedures are necessary.

A specific procedure that can be used to formally verify states of a SPFA has been proposed by Crocker (CROC77). This procedure examines the execution of a SPFA machine and refers to a state change as a state delta. These state deltas are then examined in relation to predefined lists. If a state delta results in an improper list being formed the execution of the SPFA machine during the delta is questioned for possible error.

The debugging function performed in the facility for a SPFA machine consists of the identification of an error and the isolation of its

causes by the data base machine architect. The data base machine architect isolates the error by verifying the SPFA machine states until the error occurs. This isolation can be at the source description level or at the micro-instruction level, if necessary, to insure the error is found.

During debugging, the data base machine architect views the actual representations of the SPFA machine. This permits all name variables and conditions in the SPFA description to be inspected. During the debugging exercise the DBMS has complete control of the SPFA machine to step it through various states. To assist the debugging function, the debugging methodology described by Finfer (FINF79) can be used to debug a SPFA machine. This methodology describes a Debugging Model and a set of procedures that can be used in the DMAD Facility.

4.5.3.2 Evaluation Process Functions

After the SPFA machine has been tested, the next process requested is the evaluation process. The objective of this process is to evaluate the SPFA machines that perform the DBMS function. The performance of a SPFA machine is determined using these functions:

(1) hardware operations identification, and

(2) SPFA performance .

The hardware operations identification function examines each SPFA description for identification of specific hardware operations. Each operation can be identified by one or more language statements. Once identified, an anlysis is performed to determine actual hardware timing for these operations. These timings may be based on implementing

the operation using specialized hardware components. The times for each operation are entered into the Realization Timing Control Array (RTCA) under separate realization assumptions and assigned to the SPFA machine prior to its execution.

The SPFA performance function evaluates the effects of a realization assumption chosen for a SPFA machine. This function assigns timing entries from the RTCA to each operation executed by the SPFA machine. As the SPFA machine executes, its assigned operation times are accumulated on the DBFEM. The SPFA machine is permitted to execute any number of times, each with a different assignment from the RTCA. Each execution is based on timings from different realization assumptions. The results of each execution are collected and analyzed for each SPFA machine. These results can then be analyzed to help choose the SPFA that meets the requirements of a specific application.

4.5.3.3 Substitution Process Functions

Once a SPFA machine has been selected, the final execution process is substitution. The objective of this process is to substitute the SPFA machine for the candidate software DBMS function. The substitution process consists of these functions:

(1) DBMS/SPFA machine identification,

(2) DBMS/SPFA machine selective integration, and

(3) DBMS/SPFA machine analysis.

The DBMS/SPFA machine identification function consists of identifying the configuration requirements of a DBMS machine and a SPFA machine. The configuration requirements of the machine are used to transform the DMAD Facility into the DBMS/SPFA machine. A software application program is defined to execute on this machine. This program can request functions from the DBMS supported on the DBMS machine. When this program requests the referenced DBMS function, the SPFA machine is automatically substituted to perform the DBMS function.

A DBMS/SPFA machine selective integration function performs the actual substitution of the SPFA machine. A virtual data base machine monitor (VDMM) can be used as the tool for this function. The VDMM is designed as part of a control module that supports a DBMS/SPFA machine execution. The VDMM permits the switching of control from the DBMS machine to the SPFA machine. The VDMM also maintains the complete set of resources needed to execute both machines. When the SPFA completes the DBMS function, the VDMM returns control to the DBMS machine.

Finally, a DBMS/SPFA machine analysis function assesses the impact of integrating a SPFA machine and a DBMS machine. This function requires:

(1) comparison of performance with and without the presence of the SPFA machine,

(2) identification of software application program problems resulting from the presence of the SPFA machine,

(3) identification of integration problems between DBMS machine and the SPFA machine, and

(4) determination of the feasibility of moving the referenced DBMS function from software to hardware.

Each of the processes described are available to users of the DMAD Facility. These processes provide a range of capabilities that can be utilized to develop a SPFA. Introduced in the next section are the set of functional modules used by each of the DMAD Facility hardware components to help perform these processes.

4.6 FLOW WITHIN THE DMAD FACILITY

4.6.1 Service Host Machine

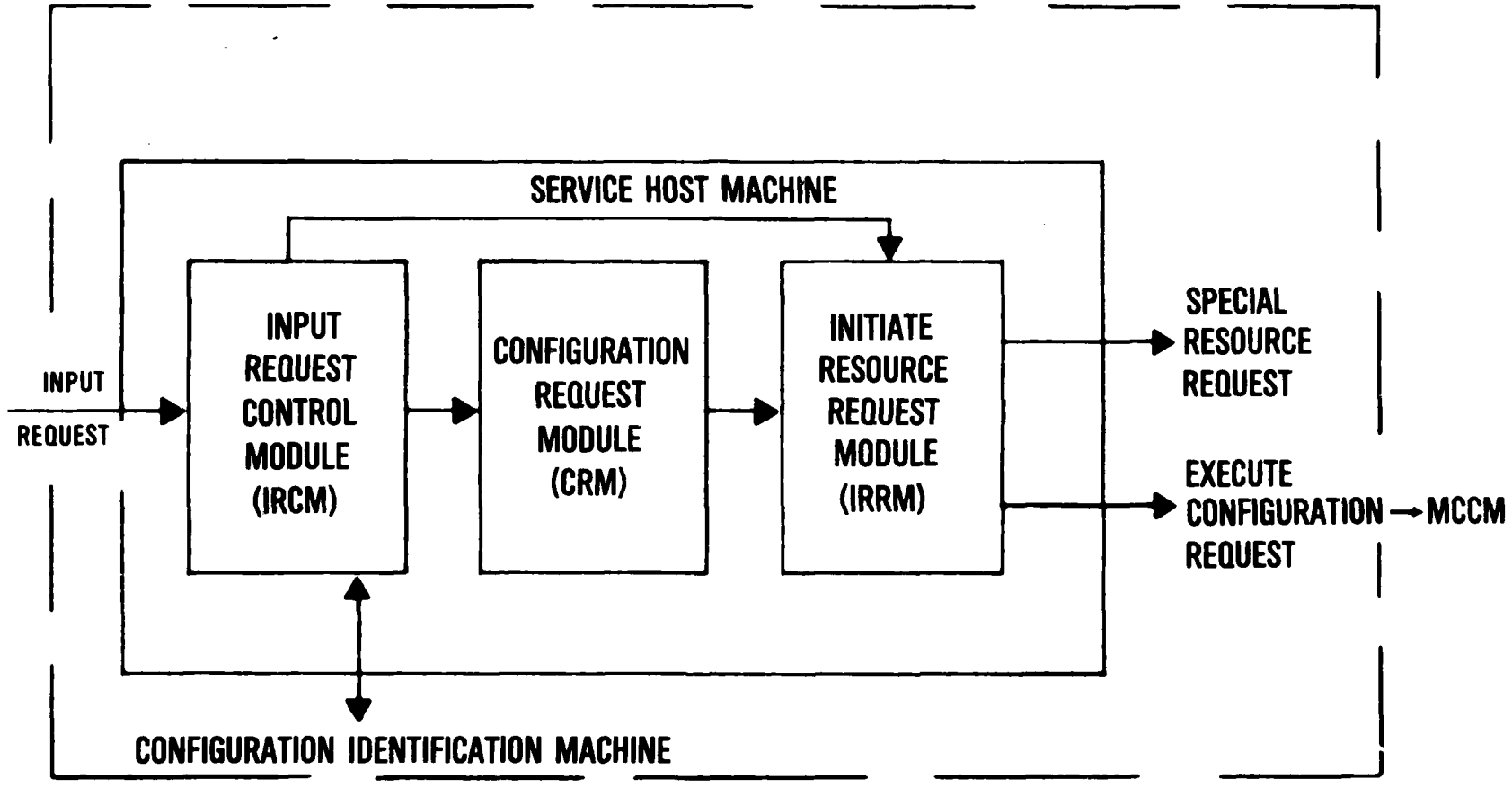
The Service Host Machine (SHM) illustrated in Figure 4-7 provides a programming environment within the facility. All inputs to the facility enter this machine. It is composed of the following modules:

(1) Input Request Control Module,

(2) Configuration Request Module, and

(3) Initiate Resource Request Module.

The flow among these modules is illustrated in Figure 4-7. Incoming requests to the SHM enter the Input Request Control Module (IRCM). This module distinguishes process requests from specific resource requests. Specific resource requests are sent directly to the Initiate



FLOW WITHIN SERVICE HOST MACHINE

FIGURE 4-7

Resource Request Module (IRRM). This module issues a direct request for service for either a local SHM resource or a network machine resource.

A process request may request one of the processes described in section 4.2. Each request needs an identification of resources for its completion. This is performed by formatting a request register from the input process request. Next, a process query is generated. Both the formatted request register and the process query are passed directly to the Configuration Identification Machine (CIM).

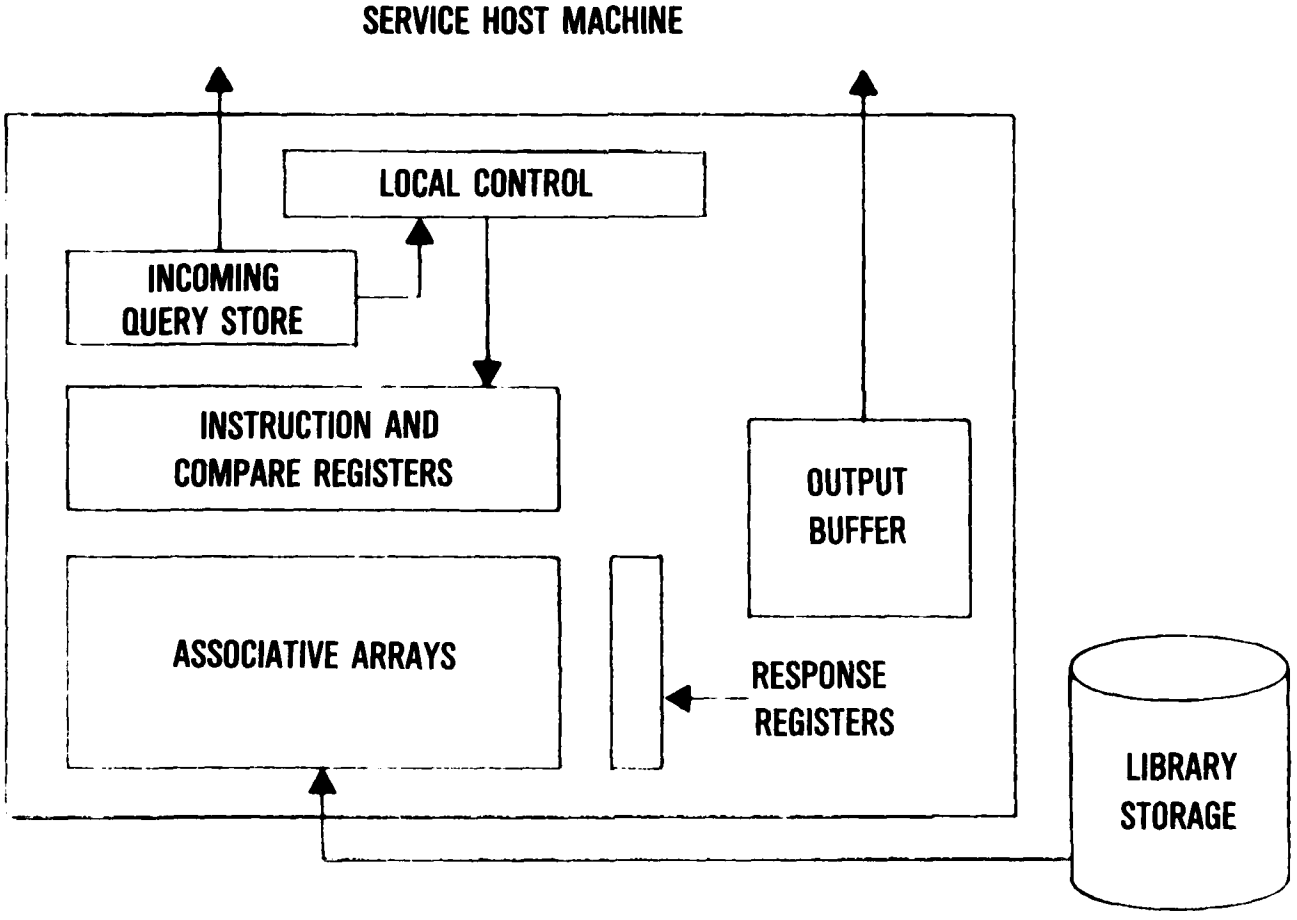
4.6.2 Configuration Identification Machine

The Configuration Identification Machine (CIM) is based on Singhania's design of an associative memory unit (SING77) and is illustrated in Figure 4-8. Queries from the Service Host Machine enter the CIM Incoming Query Store. A local control unit uses this input to load its instruction and compare registers. Next, the associative array is loaded. Two arrays are available on the CIM for process queries. These arrays are the DMCA and the RTCA.

The DMCA, illustrated in Figure 4-9, is used to identify the configuration requirements for a machine to execute in the DMAD Facility. Each entry in the DMCA identifies one of two types of machines.

(1) The first is a set of machines that have been emulated and can execute on the DBFEM. These machines support a data base management capability and are referred to as DBMS Machines, and

(2) The second type of machines are the SPFA's. These machines perform a data base management function designed in a new hardware architecture and are referred to as SPFA machines.



FLOW WITHIN CONFIGURATION IDENTIFICATION MACHINE

FIGURE 4-8

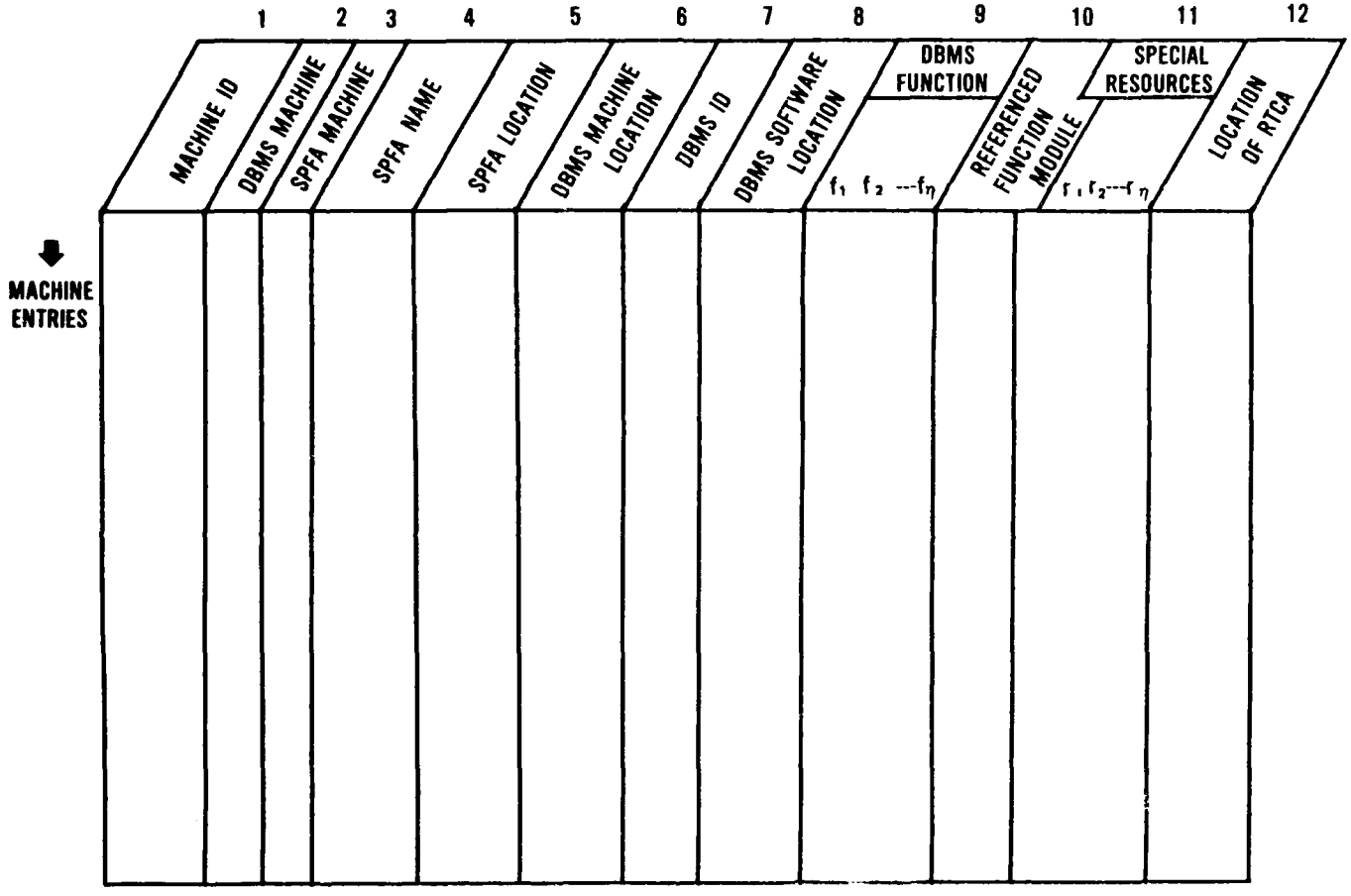


ILLUSTRATION OF A DATA BASE MACHINE ARCHITECTURE CONFIGURATION ARRAY (DMCA)

FIGURE 4-9

Each entry in the DMCA is composed of the following fields:

Field Number	Field Name
1	Machine Identification
2	DBMS Machine Entry
3	SPFA Machine Entry
4	SPFA Name
5	SPFA Location
6	DBMS Machine
7	DBMS Identification
8	Referenced DBMS Location
9	DBMS Functions
10	Referenced Function Modules
11	Special Resources
12	Location of Realization Timing Control Array

Fields 1,2 and 3 identify and classify the machine entry. If a DBMS machine is identified, its machine loadable location is identified by Field 6. Next, a data base management system supported by the DBMS machine is identified in Field 7. The location of the DBMS is defined in Field 8. A DBMS function that is supported by the DBMS is identified in Field 9. This function is called the referenced DBMS function. The software modules that perform interfaces to the referenced DBMS function are defined in Field 10. Finally, Field 11 defines special resources needed to support execution of the referenced DBMS on the DBMS machine.

If a SPFA machine is identified, its machine loadable location is identified in Field 5. Any special resources needed for its execution are defined in Field 11. Finally, Field 12 identifies the location of the SPFA machine's Realization Timing Control Array.

The RTCA, illustrated in Figure 4-10, is used to store a set of characteristics that comprise one or more realization assumptions. Each realization assumption can define a set of hardware characteristics that can be used to implement the SPFA machine's operations. Column 1 of the RTCA is used to identify a set of operation groups that are required during execution of a SPFA. Other columns, as needed, assign times to these operations based on different realization assumptions. Variations in these realization assumptions permit a DBMA to asses the performance of having different hardware characteristics complete the operation identified. For instance, one assumption may choose a fast hardware technology at a high cost to perform an operation, while another may choose a slower hardware technology B at lower cost for the same operation. These assumptions are used to make an overall performance and evaluation of the SPFA.

Once a query of the DMCA and/or the RTCA is completed, the configuration resources identified are returned to the Service Host Machine (Figure 4-7). The Input Request Control Module passes this set of resources to the Configuration Request Module (CRM). The CRM generates a configuration request using these resources. If the process is to create a SPFA, the programming environment on the SHM may be utilized. If the DBFEM is required, an execution configuration request is generated. The Initiate Request Module issues an execution configuration request to the Master Configuration Control Machine (MCCM).

CHARACTER-ISTICS / SPFA OPERATIONS	REALIZATION ASSUMPTION (RA1)	REALIZATION ASSUMPTION (RA2)	. . . . .	REALIZATION ASSUMPTION (RA n)

ILLUSTRATION OF REALIZATION TIMING CONTROL ARRAY (RTCA)

FIGURE 4-10

4.6.3 Master Configuration Control Machine

The flow using the Master Configuration Control Machine (MCCM) is illustrated in Figure 4-11. The MCCM has the following modules:

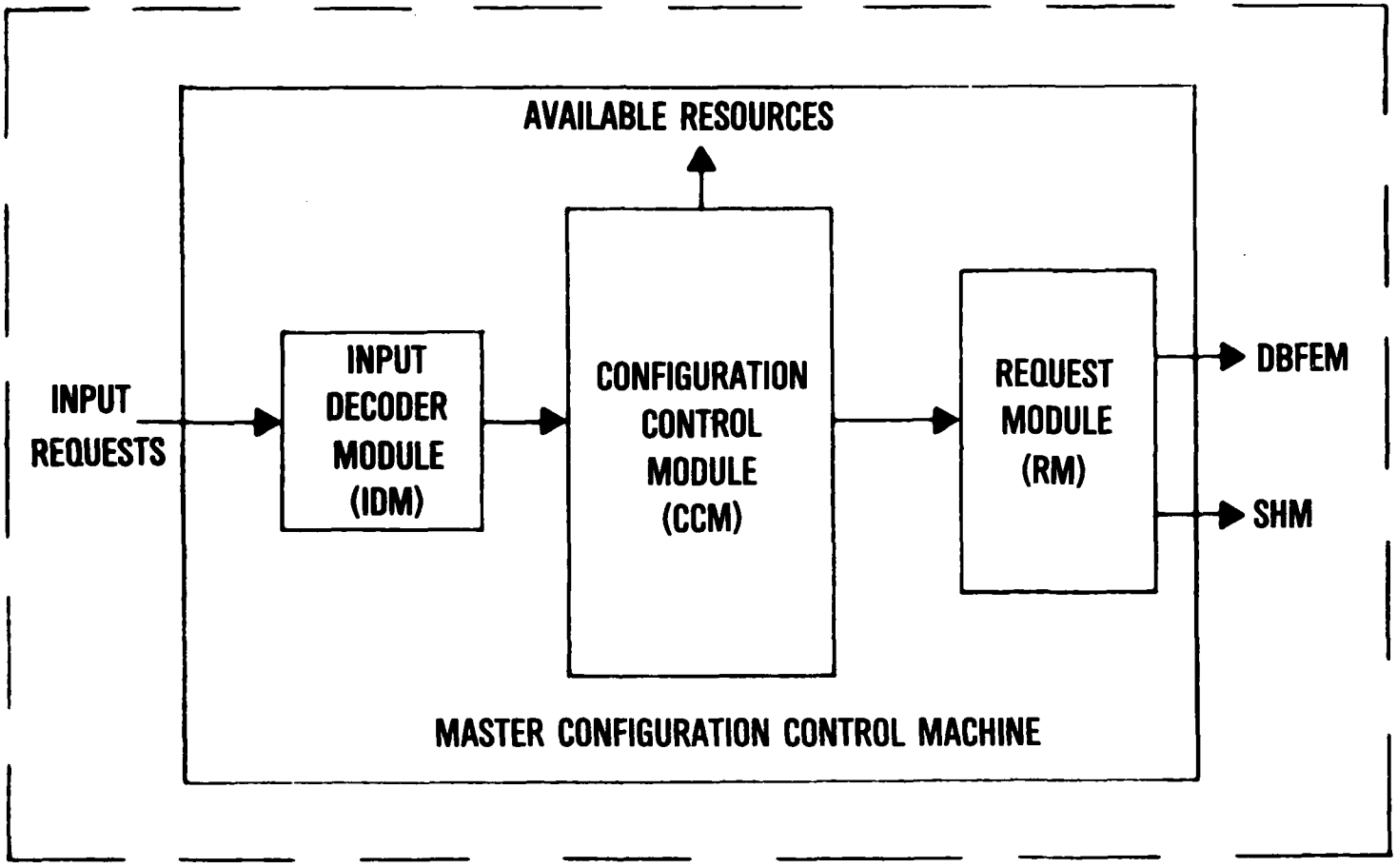
(1) Input Decoder Module,

(2) Configuration Control Module, and

(3) Request Module.

Requests to the MCCM enter the Input Decoder Module (IDM). Two types of requests to this machine are possible. These requests may be initiated from either the SHM or from the DBFEM. The results from the SHM are execution configuration requests and the requests from the DBFEM are special resource requests. The MCCM IDM determines the request type and passes control to the Configuration Control Module (CCM).

The CCM builds an Execution Request Block (ERB) for each execution configuration request. Each ERB identifies configuration resources and special resources needed for execution. All resources must be available before a request is made to configure the DBFEM. If all resources are available the ERB is included as part of an execution request to the DBFEM. The configuration resources are used to configure the DBFEM as the requested machine. Each special resource assigned to that machine can be requested during execution by issuing a special resource request. This request enters the MCCM IDM (Figure 4-11) where the request type is determined and control is passed to the Configuration Control Module. The CCM determines if this request is valid for an active machine in the facility. If the request is valid, a request for the resource is issued to the Service Host Machine.



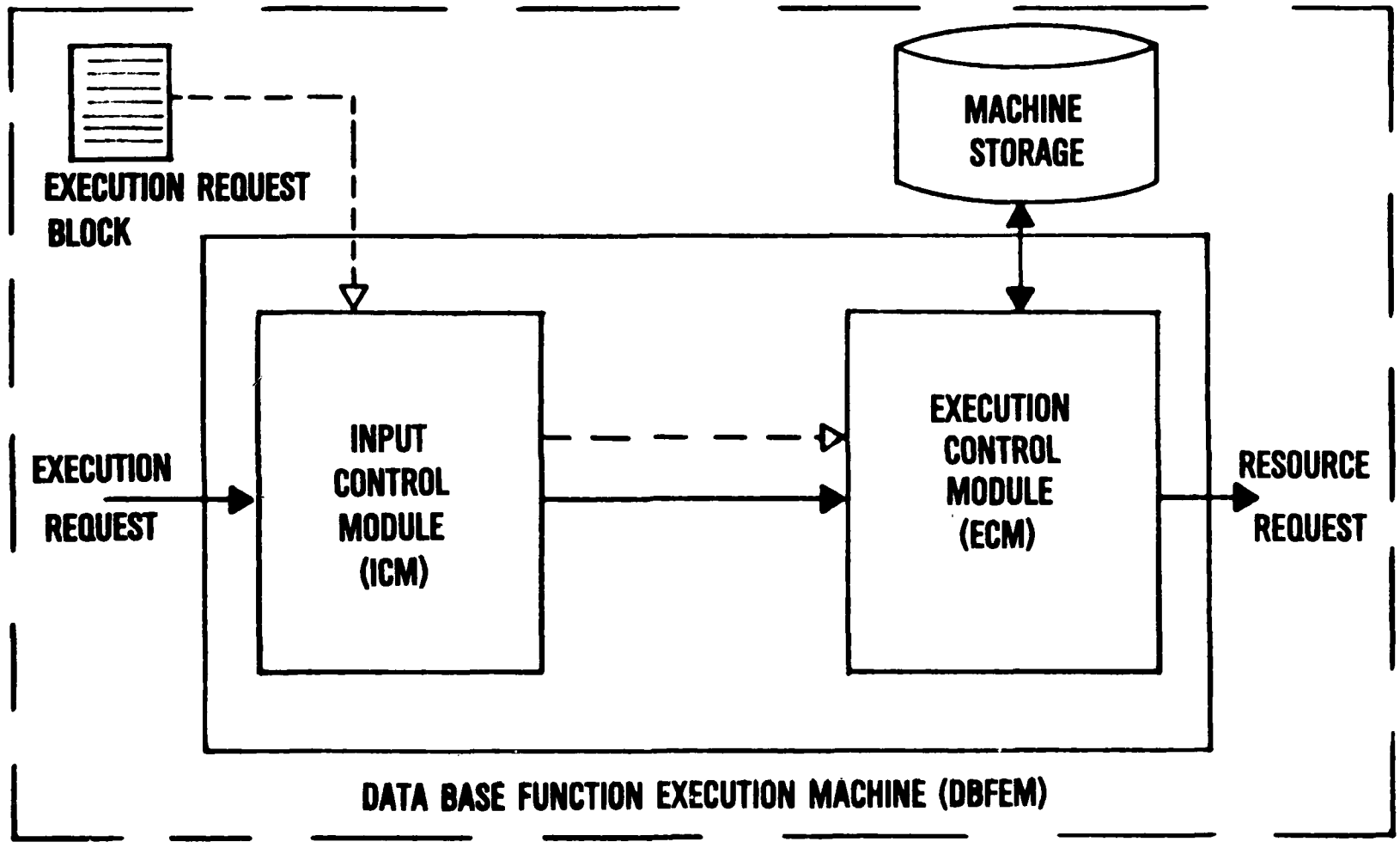
FLOW WITHIN MASTER CONFIGURATION CONTROL MACHINE

FIGURE 4-11

4.6.4 Data Base Function Execution Machine

The flow on the Data Base Function Execution Machine is illustrated in Figure 4-12. Inputs to the DBFEM enter the Input Control Module. This module determines if a new machine is being requested or if an existing machine needs to be restored. For new machines, this module determines if a DBMS, SPFA, or DBMS/SPFA machine is to be configured. Control then passes to the Execution Control Module (ECM). The ECM actually configures a new machine or re-initializes an existing machine using data from the Execute Request Block.

Illustrated in Figure 4-13 a Control Module (CM) is identified and initialized for each machine type. The Control Module loads from machine storage the descriptions of either DBMS, SPFA or DBMS/SPFA machines. Each DBMS machine is configured with system support software, a DBMS and a set of application programs. A SPFA machine is configured with a machine driver. The driver defines, for each SPFA, a function request along with a set of needed inputs. This request determines the execution sequence of the SPFA machine. Once configured, each SPFA then executes on the DBFEM in a manner consistent with its architectural description. During this execution, the data base machine architect can test the SPFA machine to insure that the sequence of operations it performs accomplishes the DBMS function it is designed for. Following testing, the evaluation process can perform an architecture evaluation of the SPFA, and timing data can be generated to help in the evaluation. Following evaluation, the DBFEM may be reconfigured as a DBMS/SPFA machine for a substitute process. The entire set of development processes can be used and reused until the SPFA development is complete.



FLOW WITHIN DATA BASE FUNCTION EXECUTION MACHINE (DBFEM)

FIGURE 4- 12

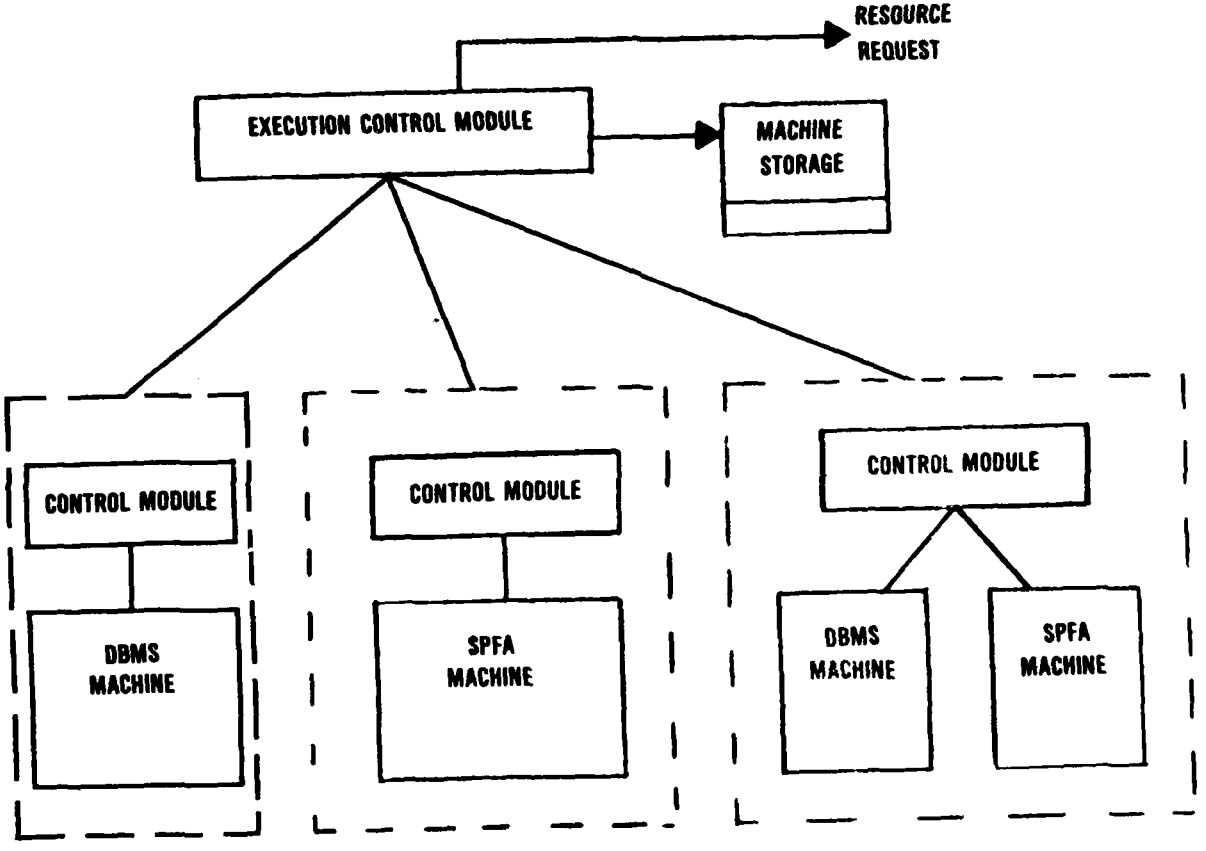


ILLUSTRATION OF EXECUTION MACHINE TYPES

FIGURE 4-13

A completed SPFA development can represent one or more SPFA machines that have been executed, have had a comparative performance data analysis performed and each is assessed as to how well they integrate into a data base management system. Data from this completed development is then used to choose if one of the SPFA's should be hardware prototyped for this specific application.

4.7 SUMMARY

Described in this chapter is a Data Base Machine Architecture Development. It is shown how the tools/components within this facility can be used to help migrate DBMS functions from software into hardware. A methodology is described as a flow of processes that permits facility users to choose candidate DBMS functions and to develop special purpose function architectures for these functions. The utilization of this facility, in this manner, can provide a unique capability for helping to develop new DBMS hardware architectures for non-numeric functions.

Described in the next chapter are specific details on utilizing the tools/components in the facility, along with a set of detailed procedures that can be used to perform the DMAD Facility development processes.

CHAPTER V

DEVELOPMENT OF SPFA'S USING THE DMAD FACILITY

5.1 INTRODUCTION

Described in this chapter are several procedures used to perform functions for each DMAD Facility process. These procedures can be performed with tools and components of the DMAD Facility. Also, described in this chapter are the functional modules required for the DMAD Facility hardware components. These modules control the flow of a user request in the facility. Selective sets of examples for each process are also described.

5.2 PROCEDURES FOR SELECT CANDIDATE FUNCTION PROCESS

Each sequential computer system that can be emulated and is capable of supporting a DBMS is referred to as a DBMS machine. Such a machine is illustrated in Figure 5-1. This machine supports a series of DBMS software modules that constitute its DBMS capability. These modules are the candidates to be moved to hardware. A DBMS machine is introduced to the DMAD Facility using the identification procedure described below.

5.2.1 DBMS Machine Identification

Each sequential computer system that is introduced to the DMAD is described using a hardware description language tool available in the facility. This description becomes the DBMS machine emulator and is entered as an available machine in the Data Base Machine Architecture Configuration Array. The DBMS functions available on this machine are

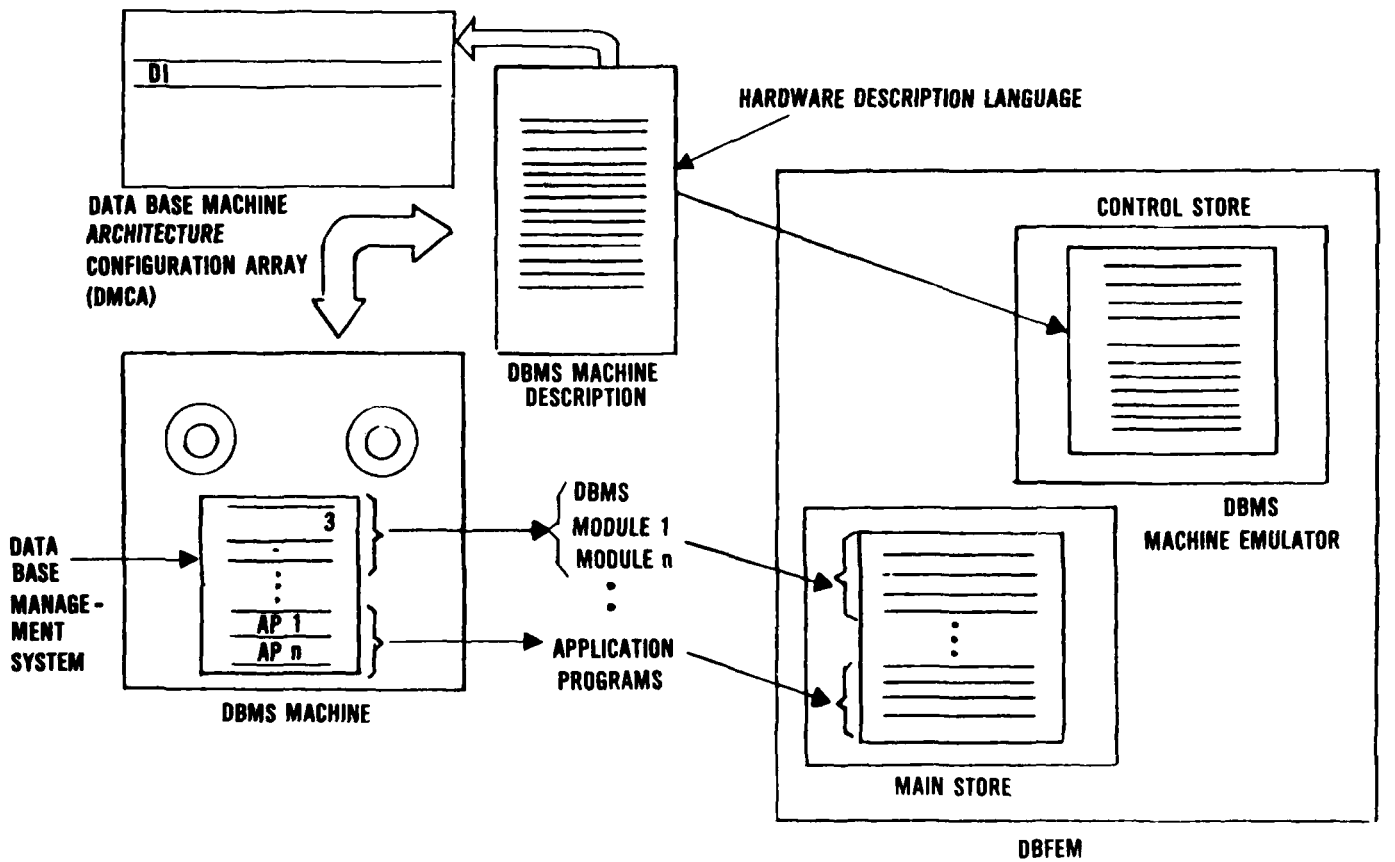


ILLUSTRATION OF DBMS MACHINE EXECUTION

FIGURE 5-1

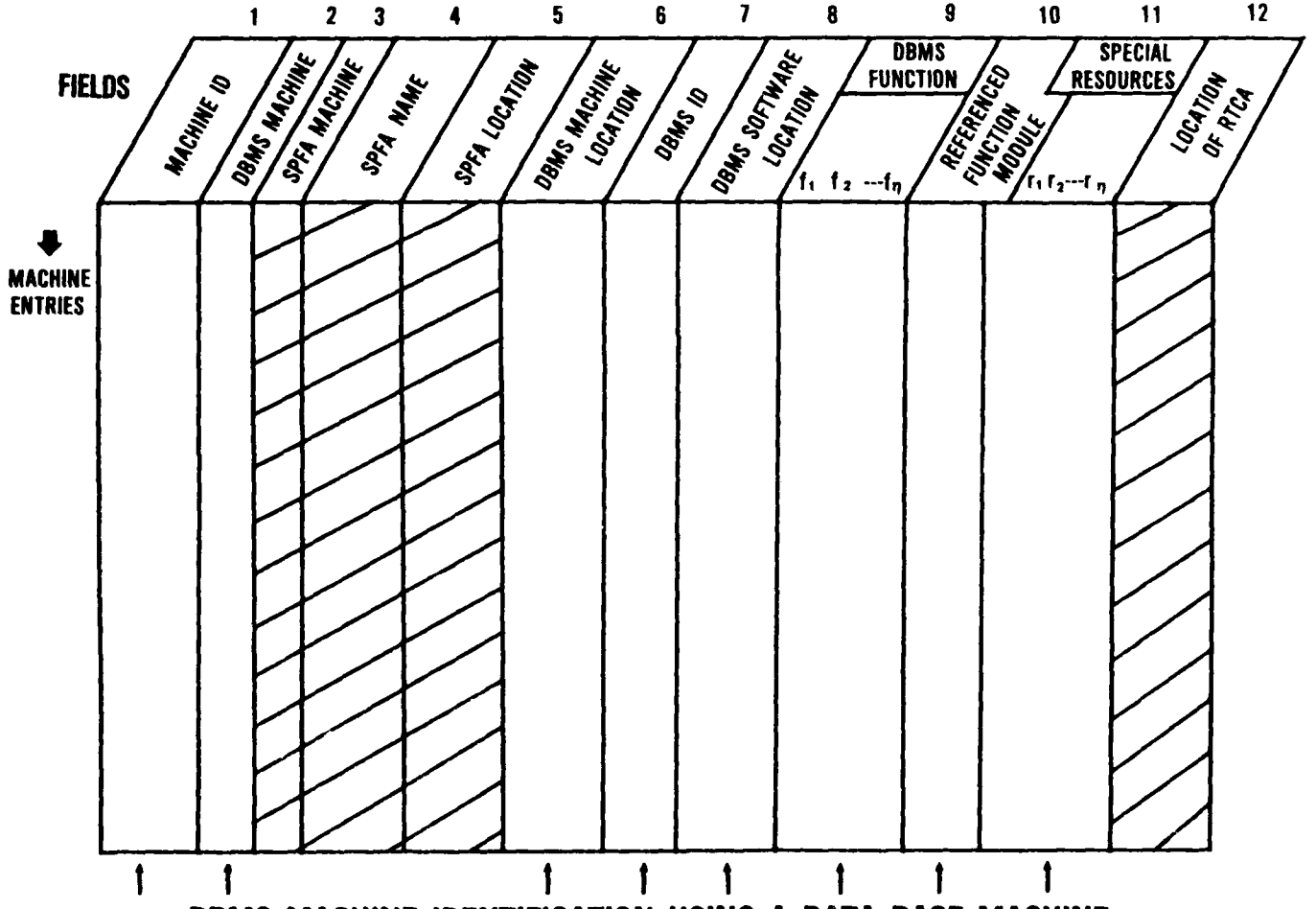
part of one or more DBMS's it supports. These functions, written as software modules, perform either data manipulation, data maintenance, user interface or application dependent procedures for a specific DBMS.

A typical DBMS machine entry in the DMCA is illustrated in Figure 5-2. This entry consists of the following fields:

Field #	Field Name	Field Description
1	Machine Identification	Identifies the unique machine entry
2	DBMS Machine	Defines entry as a DBMS machine
6	DBMS Machine Location	Identifies the loadable version of the DBMS machine in machine storage
7	DBMS Identification	Identifies a DBMS that executes on the DBMS machine
8	DBMS Location	Identifies the location of this DBMS in configuration file
9	DBMS Functions	Identifies DBMS functions supported by the DBMS
10	Referenced Function Modules	Identifies modules for each referenced function
11	Special Resources	Identifies special resources needed during execution of the DBMS machine

5.2.2 DBMS Machine Execution

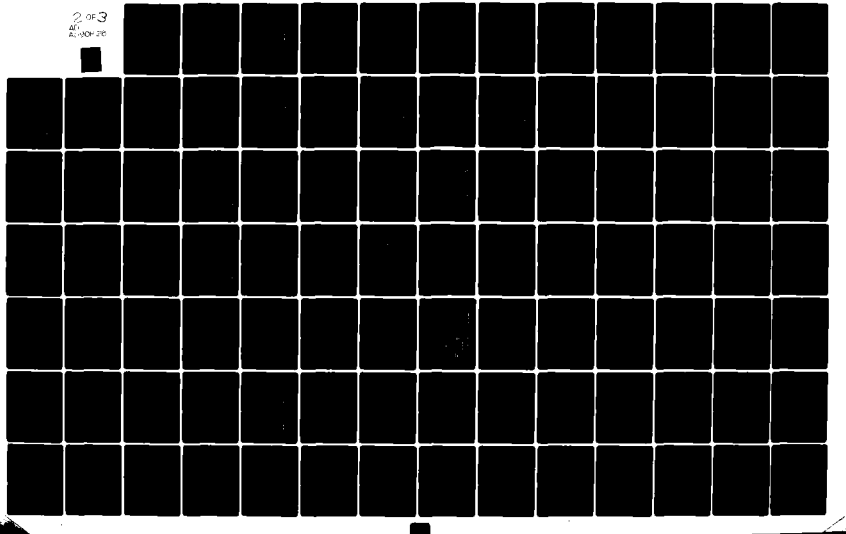
Once the DBMS machine is entered in the DMCA it is available for execution in the facility as illustrated in Figure 5-1. The same set of software modules that execute on the real DBMS machine computer system now execute on the emulated DBMS machine. These modules execute in a manner consistent for which they were originally written.



DBMS MACHINE IDENTIFICATION USING A DATA BASE MACHINE ARCHITECTURE CONFIGURATION ARRAY

FIGURE 5-2

AD-A090 826 ROME AIR DEVELOPMENT CENTER GRIFFISS AFB NY F/G 9/2
THE SPECIFICATION OF A DATA BASE MACHINE ARCHITECTURE DEVELOPME--ETC(U)
JUL 80 R A LIUZZI
UNCLASSIFIED RADC-TR-80-256 NL



With this capability, the DMAD facility can be utilized for data base management development in the following ways:

(1) First, new data base management systems can be developed to execute on the emulated machine. This permits a user to evaluate how well the new DBMS will perform on the emulated machine without having the machine physically present.

(2) Second, to insure compatibility during upgrades of the real computer system. Upgrades to the real computer system can be made with assurances that the data base management system and its supporting application programs can still execute on the emulated computer system. Thus, this sytem can still be made available to users if their application programs are not completely compatible with the upgraded real system.

(3) Third, to help select candidate DBMS functions to be moved from software to hardware.

The first two are traditional utilizations of emulation and data base management system development. The selection of candidate DBMS functions illustrates a further capability that is provided by the tools/components of a DMAD Facility.

5.2.3 Candidate DBMS Function Selection

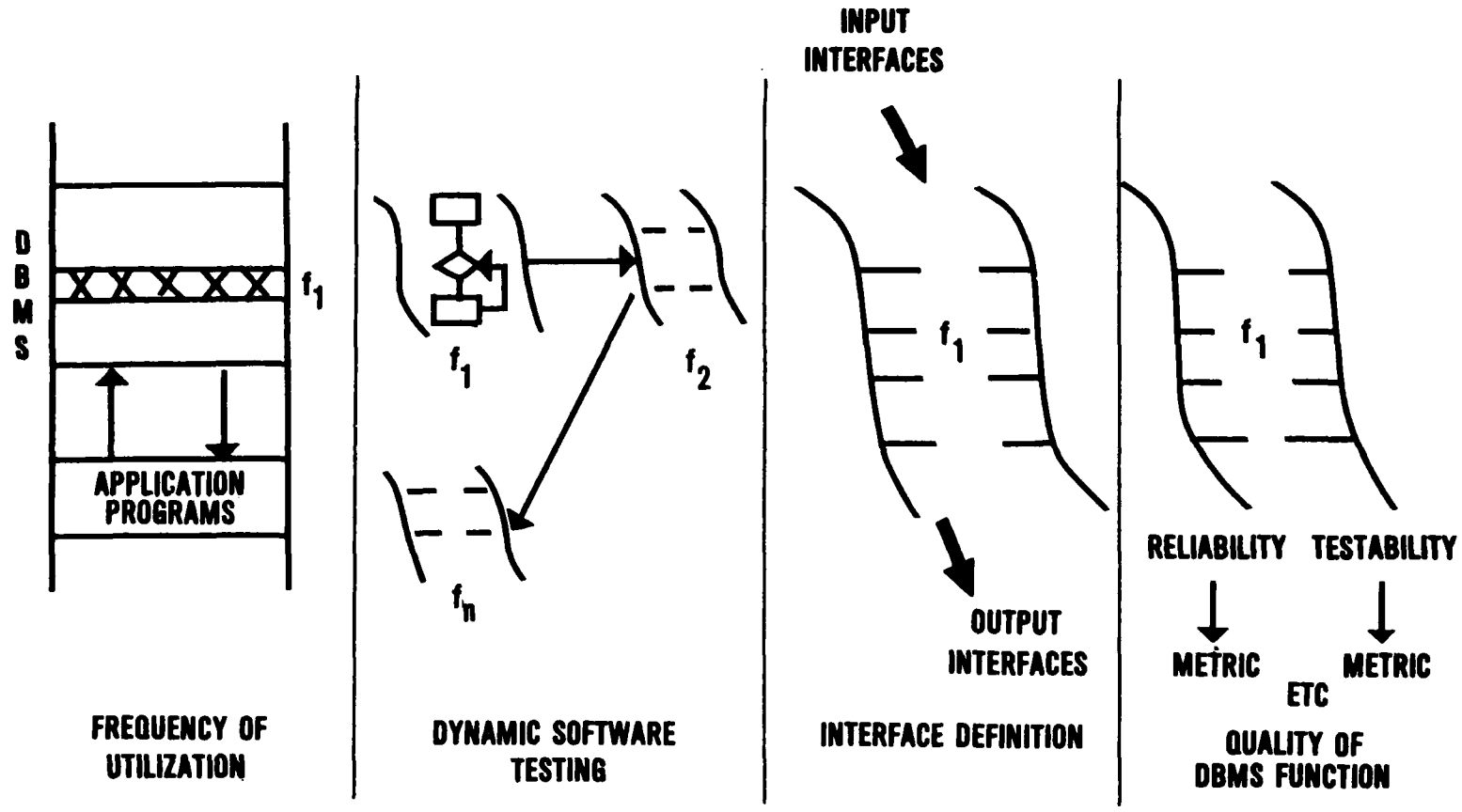
A candidate function can be chosen in many ways using the facility. Some of these are illustrated in Figure 5-3. First, a candidate DBMS software function may be selected by collecting statistics on its frequency of utilization (i.e. times called, processing time, etc.).

During execution of a DBMS machine, if these statistics indicate high utilization of a specific module, it may be selected as a candidate function. Once a candidate is selected, specific interfaces for the software function need to be defined. This includes identifying the set of inputs and outputs to the function. Next, dynamic testing of the software module function is conducted to follow execution paths and help define its interfaces. Finally, software quality metrics for each module can be established. Each of these types of data collection is discussed below.

5.2.3.1 Data Collection

Data collection on utilization of a specific function can be obtained by establishing breakpoints at the entrance to a software module each time a specific function executes. The utilization rate can be determined from the number of times the breakpoint is encountered. Another procedure is to utilize a facility performance monitoring tool to identify frequency of calls for a DBMS function. Such a capability can be provided as part of the hardware description language used to describe the DBMS machine emulator. For instance, the SMITE hardware description language provides such a performance capability (TRW79) and is used to count the number of times an instruction is encountered. Both the breakpoint procedure or the performance monitoring capability represent the types of procedures that can be used in the facility to establish utilization statistics for candidate functions.

Once utilization data is collected, the facility can be used to examine the execution path of highly called functions. This path can be examined by actually stepping the execution of the function, instruction by instruction. This is done by dynamically testing the software



PROCEDURE TO IDENTIFY DBMS FUNCTIONS USING DMAD FACILITY

FIGURE 5-3

module to establish interfaces required to and from the function and complexity of the function in relation to other functions within the DBMS.

Once a complete set of this type of data for the function has been collected, using the facility, another type of analysis may be used to help decide if this is a candidate. This analysis is based on the overall quality improvement that may be obtained by moving the function to hardware. For instance, will the move of this function to hardware increase DBMS system performance but at the same time decrease the systems portability of reliability?

Questions such as these may be examined by using the facility to help establish metrics for specific quality factors concerning the DBMS module software. These metrics can be computed for several factors including reliability, maintainability, flexibility, etc. of the DBMS software modules using the procedures developed in a study by Cavano and McCall (CAVA78). If the replacement of specific modules by SPFA's has the potential for improving the quality of the DBMS, then, those modules may become candidate functions.

In summary, the choice of specific DBMS functions to be moved may be based on criteria that add to its utilization, performance and complexity, and quality within the system. The facility can be used to help compute and establish a wide range of data from which a DBMS can select candidate functions. Once one or more candidates are selected, a DBMA needs to examine how and if these functions can be performed by special purpose function architectures. This examination can also be conducted using the facility with the SPFA development processes.

5.3 PROCEDURES FOR THE CREATE PROCESS

Each SPFA to be developed in the DMAD Facility must first be originated by a create process request. This request requires a procedure to build a DMCA machine entry for a new SPFA. The attributes assigned to this entry are used to configure the DMAD Facility for execution of this machine. A SPFA description development function is required to describe each new SPFA. This function can be performed by the procedure given below.

5.3.1 SPFA Description Development

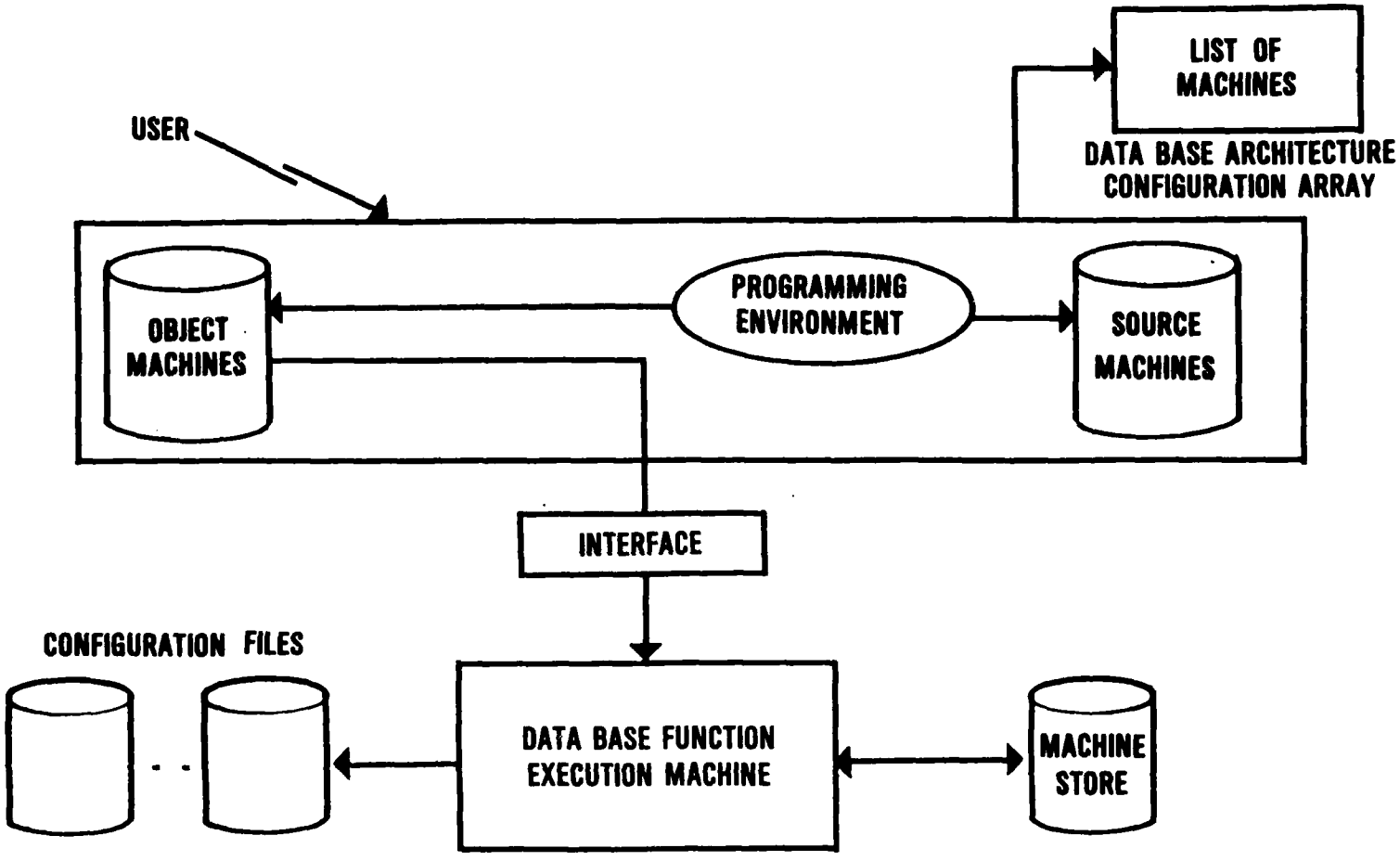
A procedure for the SPFA description development function illustrated in Figure 5-4 requires:

(1) Develop SPFA description,

(2) SPFA compiliation, and

(3) SPFA correction.

A specialized programming environment on the Service Host Machine is available to develop SPFA descriptions. These descriptions are compiled, debugged and edited within this environment. Each SPFA is described in a hardware description language (HDL). This includes identifying the machine representation of the SPFA in terms of registers, interconnections, flow of information and specific operations. Both control and concurrency dependencies among SPFA operations are described. In addition, the SPFA must accept as input the same interfaces identified by the select candidate function process. Next, a SPFA compiliation procedure translates this SPFA description into a source and object file. The source file is then debugged within the programming environment to eliminate source



DMAD FACILITY ENVIRONMENT FOR DEVELOPING MACHINES

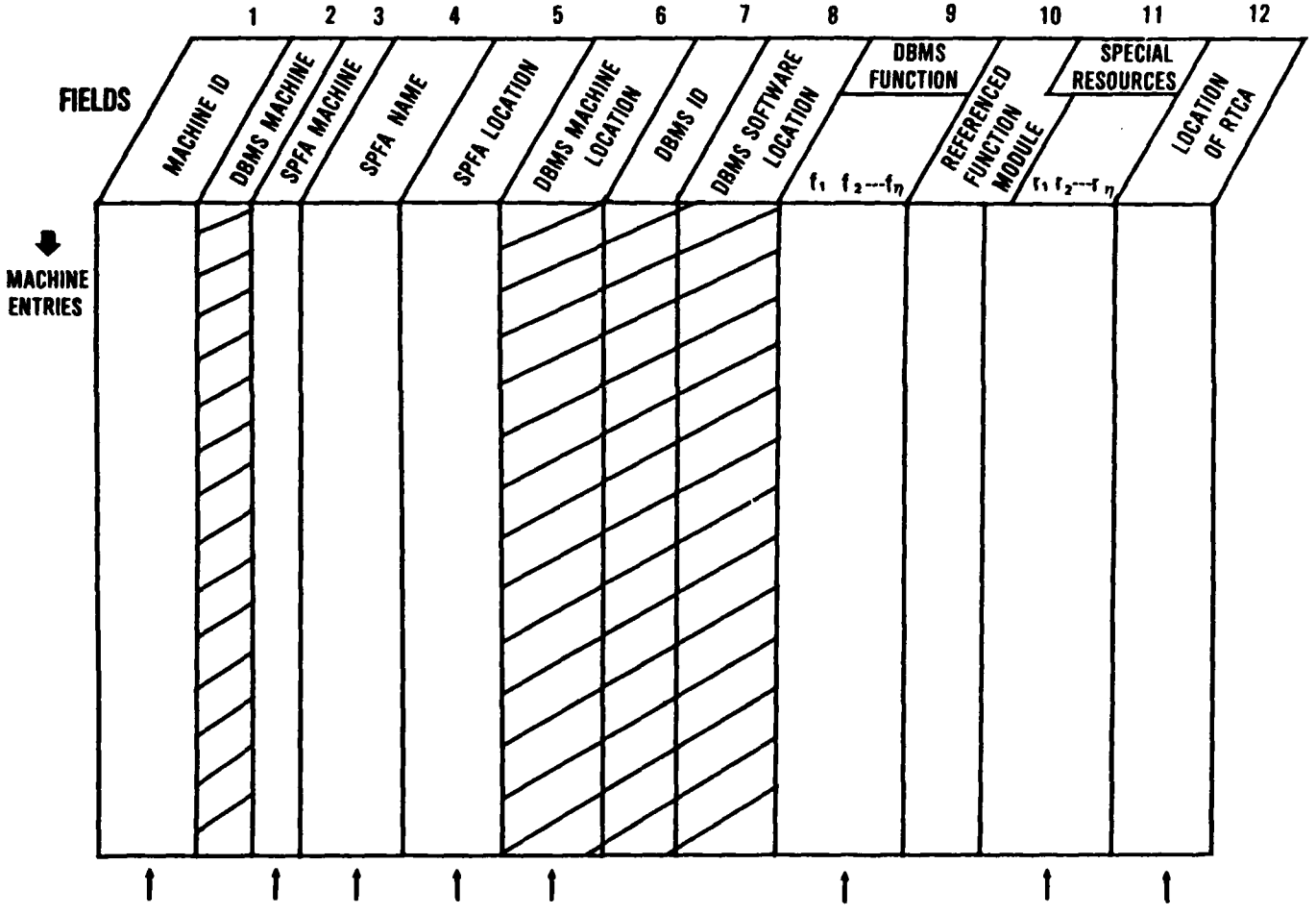
FIGURE 5-4

programming errors. If errors are identified, the SPFA description is modified and recompiled. This procedure continues until a correct SPFA is described. This machine is entered in the DMCA. The SPFA compilation also produces an object file that consists of a set of micro-instructions. These object instructions are transferred to a machine storage. These micro-instructions are loaded on the DBFEM and are used to help transform the facility into a SPFA machine.

5.3.2 SPFA Introduction

Once a SPFA has been described it can be entered in the Data Base Machine Architecture Configuration Array (DMCA). A SPFA introduction function is performed by creating an entry for the SPFA in the DMCA, as illustrated in Figure 5-5. The fields needed to introduce a SPFA machine in the DMCA are also shown in Figure 5-5. These fields are defined as follows:

Field (1) Machine Identification: This field assigns a unique machine identification number to the machine in the DMAD Facility.



SPFA MACHINE IDENTIFICATION USING DATA BASE MACHINE ARCHITECTURE CONFIGURATION ARRAY

FIGURE 5-5

Field (3) SPFA Machine: This field identifies this entry as a SPFA machine.

Field (4) SPFA Name: This field assigns a name to each version of a SPFA.

Field (5) SPFA Location: This field identifies the location of the DBFEM loadable version of the SPFA machine.

Field (9) DBMS Functions: This field identifies which of the pre-defined set of $f_1, f_2 -- f_n$ DBMS functions the SPFA performs. These functions can include a DBMS basic service function, a DBMS user function, DBMS functions, or a DBMS application function.

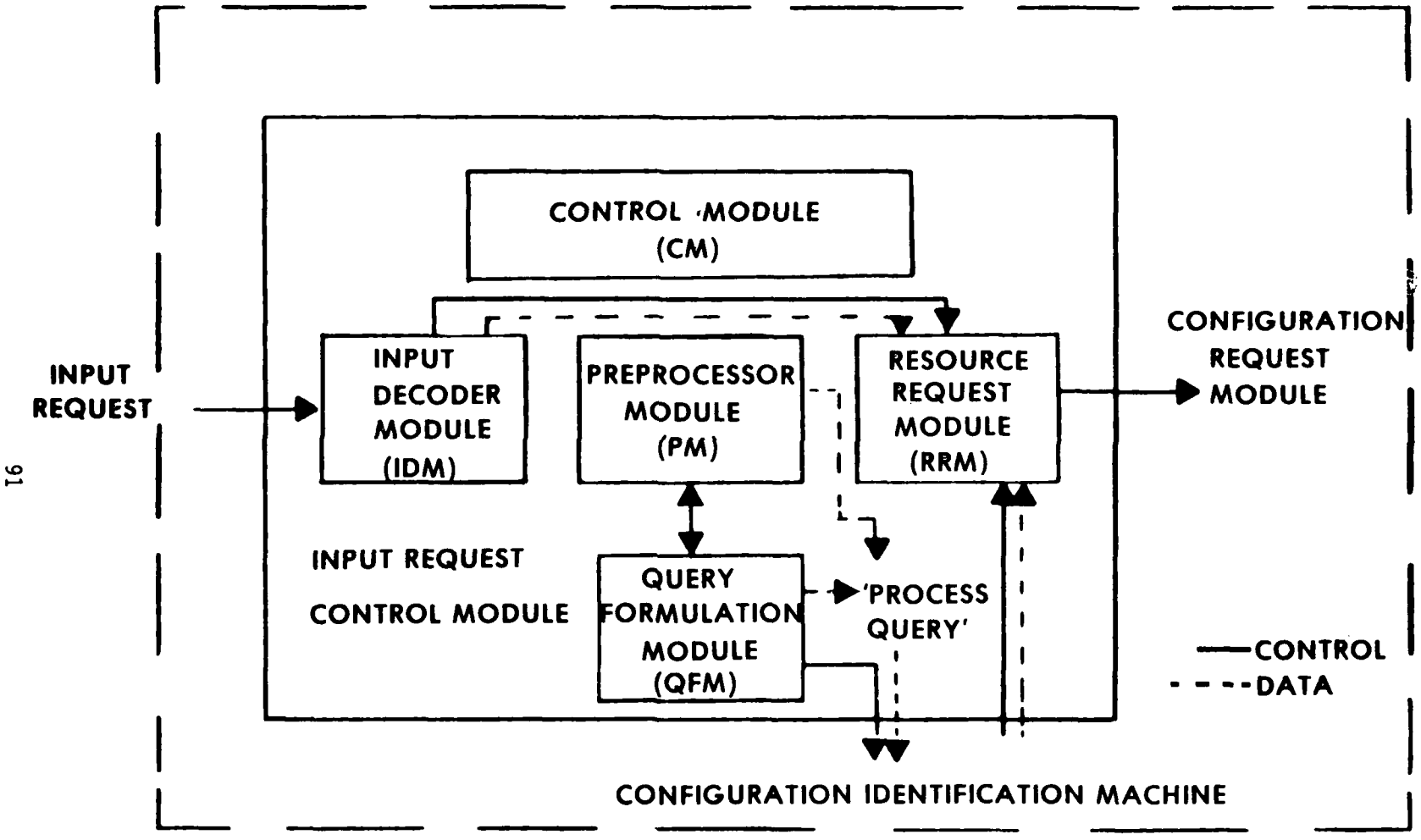
Field (11) Special Resources: This field identifies special resources required to assist the SPFA machine execution.

Field (12) Realization Timing Control Array (RTCA): This field identifies the location of the RTCA for the SPFA machine entry.

Both the select candidate function process and the create process require use of the facility's Service Host Machine and the Configuration Identification Machine. The functional flow on these machines is now described.

5.4 SERVICE HOST MACHINE

A DMAD Facility request enters the Service Host Machine Input Request Control Module (Figure 5-6).



FLOW WITHIN SERVICE HOST MACHINE INPUT REQUEST CONTROL MODULE

FIGURE 5-6

5.4.1 Input Request Control Module

The SHM Input Request Control Module, shown in Figure 5-6, is composed of:

(1) Control Module,

(2) Input Decoder Module,

(3) Pre-processor Module,

(4) Query Formulator Module, and

(5) Resource Request Module.

The Control Module controls the flow of each input request. Each input request enters the Input Decoder Module. The decoder distinguishes process and special resource requests. Process requests are directed by the Control Module to the Pre-processor Module, while resource requests are directed by the Control Module to the Resource Request Module (RRM). The RRM uses this input to formulate a request type for the Configuration Request Module.

Each input process request is in the form of a process description language (PDL). The process description language varies for each process request. The PDL can consist of the following:

(1) a process request type,

(2) a machine identification,

(3) a SPFA name,

(4) a DBMS identification,

(5) a DBMS function,

(6) special resources, and

(7) realization timing control array assignments.

The Pre-processor Module formulates an input to the CIM from the PDL. The format of this input is illustrated in Figure 5-7. It consists of the following:

(1) Request ID: Each process request is assigned a unique identification. This ID is used to identify active processes in the DMAD Facility.

(2) DMAD Process: The process type is identified. A DMAD process consists of one of the following:

Process Select Candidate - p_1

Process Create - p_2

Process Test - p_3

Process Evaluation - p_4

Process Substitution - p_5

(3) Machine Identification: This field identifies a machine and a machine type.

(4) SPFA Name: This field identifies a unique name for a SPFA machine type.

(5) DBMS Identification: This field identifies a specific DBMS that is supported by a DBMS machine.

(6) DBMS Functions: This field identifies a DBMS function.

(7) Special Resources: This field identifies additional special resources required to execute the defined machine, other than identified by its entry in the DMCA.

(8) Realization Timing Control Array Assignments: This field identifies assignments from the RTCA.

FIELDS

REQUEST ID	DMAD PROCESS	MACHINE IDENTIFICATION		SPFA NAME	DBMS ID	DBMS FUNCTIONS	SPECIAL RESOURCES	RTCA ASSIGN-MENTS
		ID	D S B P M F S A					
ID1	p2	s1	01	MERGE	....		f1	r2	....

EXAMPLE

SAMPLE FORMAT OF INPUT TO CONFIGRATION IDENTIFICATION MACHINE

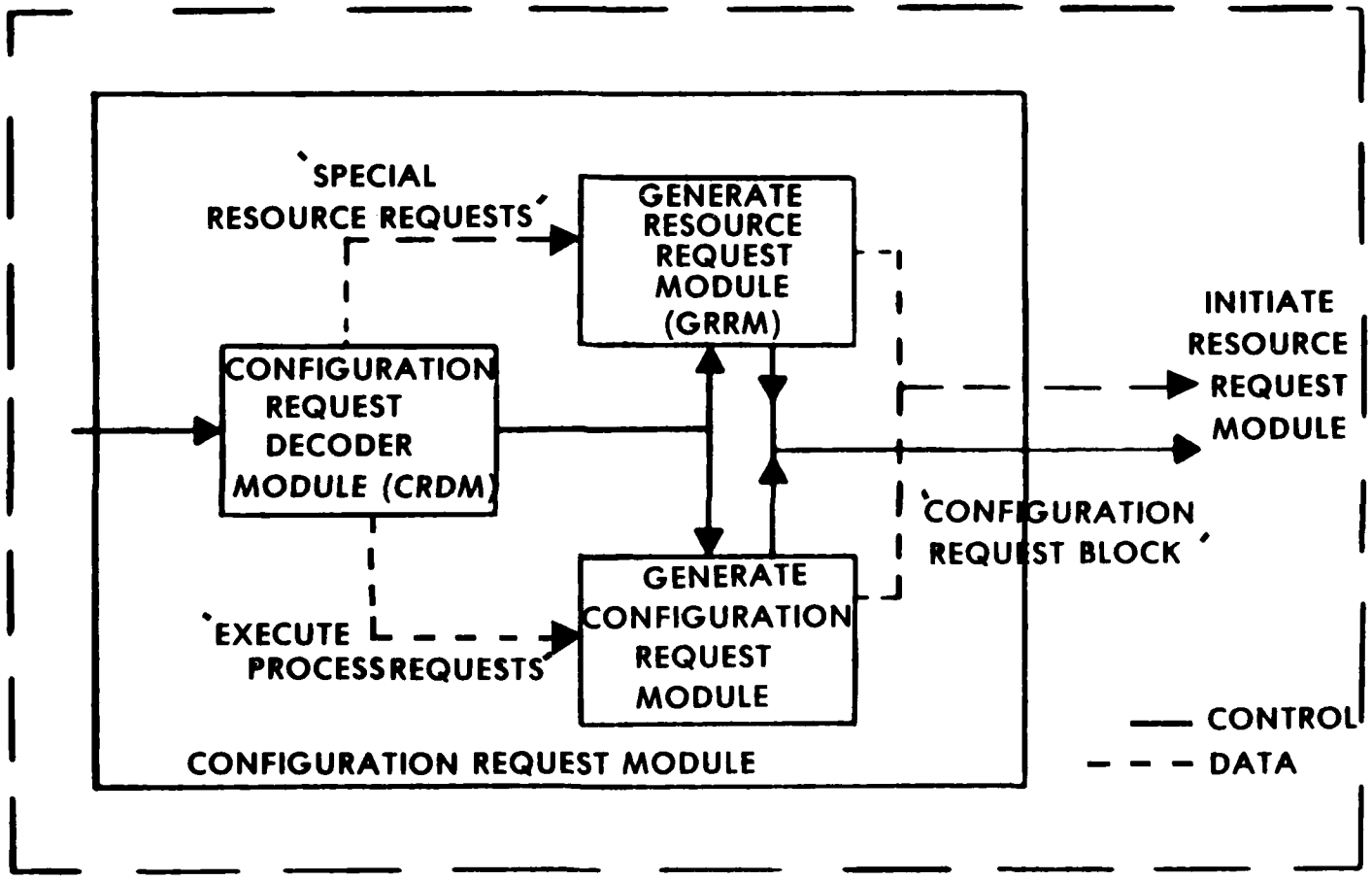
FIGURE 5-7

Once this input is formatted, a process query must be generated. The Query Formulation Module generates a process query (Figure 5-6) that is used to identify machines and configuration resources, from the DMCA, for the type of process requested. When a process query has been generated it is passed, along with the formatted input, to the Configuration Identification Machine. The query identifies, from the DMCA, a set of resources used to configure a machine in the facility. The set of defined resources are returned to the Input Request Control Module (Figure 5-6). The Control Module directs the resources to the Resource Request Module (RRM), which uses the resources to formulate input to the Configuration Request Module.

5.4.2 Configuration Request Module

The Configuration Request Module (Figure 5-8) develops a Configuration Request Block (CRB) for all request types in the facility. The CRB defines the resources to support a resource request or an execute request. A Configuration Request Decoder Module (CRDM) distinguishes the process request type. For instance, a process may require tools from the SHM programming environment or execution of a machine using the DBFEM. A select candidate process may request tools to develop a DBMS machine or require execution of a DBMS machine. A create process requires tools from the SHM programming environment to develop a description of a SPFA. Test, evaluate and substitute processes require executing a SPFA machine.

A Generate Resource Request Module (GRRM) formulates a resource request for tools from the SHM programming environment while the Generate Configuration Request Module (GCRM) (Figure 5-8) formulates an



FLOW WITHIN SERVICE HOST MACHINE CONFIGURATION REQUEST MODULE

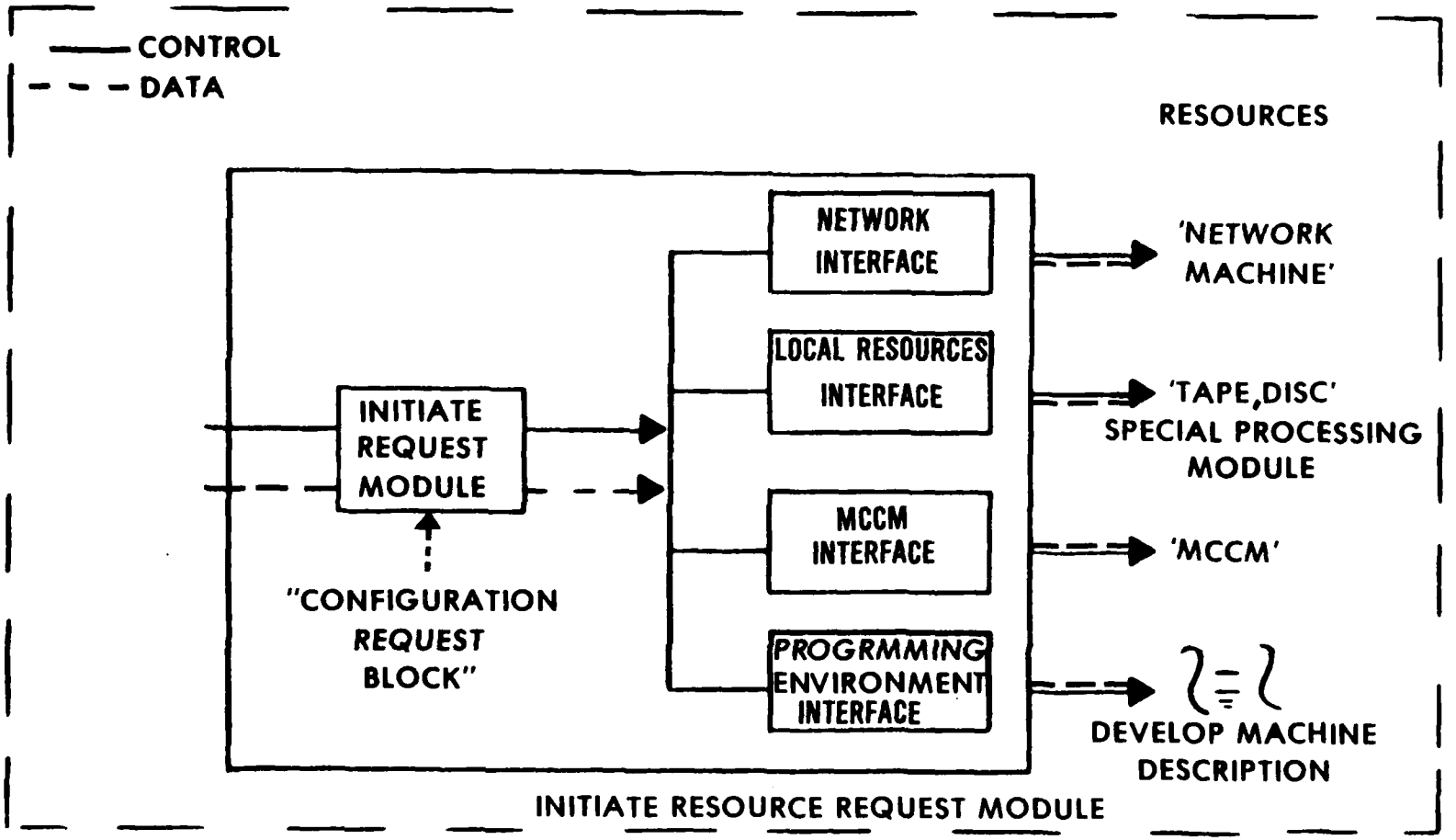
FIGURE 5-8

execution configuration request. Each request is passed to the Initiate Resource Request Module.

Special resource requests can also be made by machines executing in the facility. Special resources may include local SHM resources, special processing modules or network machines. Local resources are tapes, discs, etc. available on the SHM. A special processing module is a module that performs a specific task for a machine during its execution. For instance, network machines may be called to provide highly specialized tools that support a SPFA machine execution. These requests also enter the Configuration Request Module where a CRB for these resources is also formulated by the GRRM. Once formulated, this request is also passed to the Initiate Resource Request Module.

5.4.3 Initiate Resource Request Module

The Initiate Resource Request Module (Figure 5-9) generates the actual resource request. This module accepts the Configuration Request Block from the Configuration Request Module. This block distinguishes the type of request. For a resource request, the Initiate Request Module directs control to the appropriate interface. These interfaces offer service to the program environment, a local resource, or network machines. Control then passes to the resource to complete the desired operation. Execution configuration requests are directed to the Master Configuration Control Machine Interface (MCCM). This interface permits control to pass from the SHM to the MCCM. Each execution configuration



FLOW WITHIN SERVICE HOST MACHINE INITIATE RESOURCE REQUEST MODULE

FIGURE 5-9

request identifies the resources needed for a machine configuration. These include:

(1) locations of machines to be executed on the DBFEM,

(2) special resources needed to support their execution,

(3) identification of a Control Module,

(4) identification of assignments from the RTCA,

(5) location of a SPFA machine driver, and

(6) other special configuration needs.

Part of the resources needed to configure a machine are identified from the process query of the DMCA on the Configuration Identification Machine.

5.5 CONFIGURATION IDENTIFICATION MACHINE

The Configuration Identification Machine (CIM) is illustrated in Figure 5-10. The CIM is composed of the following:

(1) Incoming Query Store,

(2) Local Control Unit,

(3) Request Register,

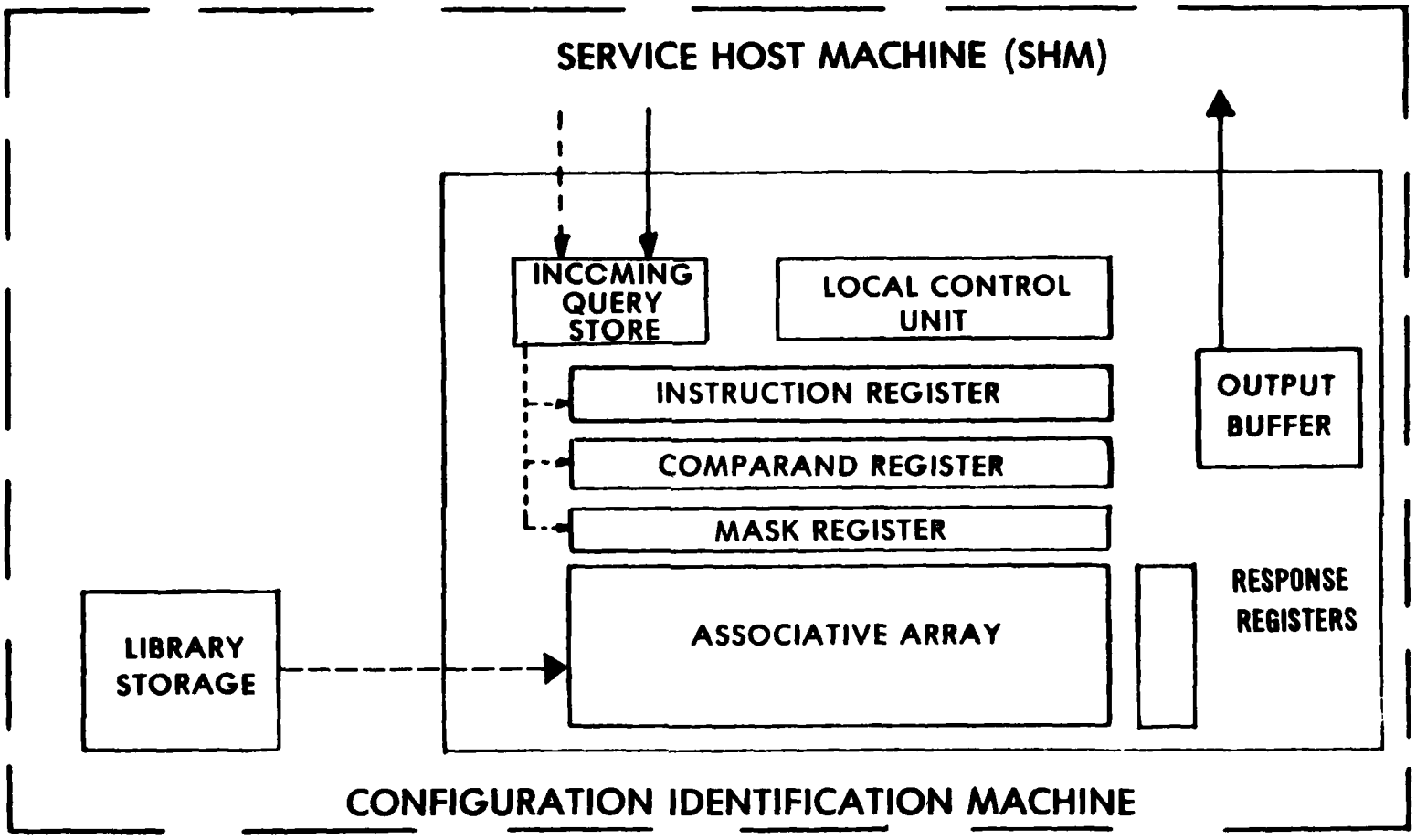
(4) Comparand Register,

(5) Mask Register,

(6) Response Register, and

(7) Word Select Registers.

Inputs to the CIM enter its Incoming Query Store. An input consists of a formatted request and a process query. The Local Control Unit (LCU) loads the Comparand Register with the formatted request. The LCU loads instruction sequence of the process query in the Instruction Register. The Local Control Unit initializes the DMCA in the



FLOW WITHIN CONFIGURATION IDENTIFICATION MACHINE

FIGURE 5-10

CIM associative array. The DMCA is interrogated to identify the appropriate machine or machines for a process request. Once a machine is identified its configuration resources are directed to an output buffer and returned to the SHM (Figure 5-6). These resources are included in the request passed to the Configuration Request Module.

5.5.1 Example SPFA Creation

An example of how a SPFA is created is described below. First, a sample process description language for a create SPFA process can be:

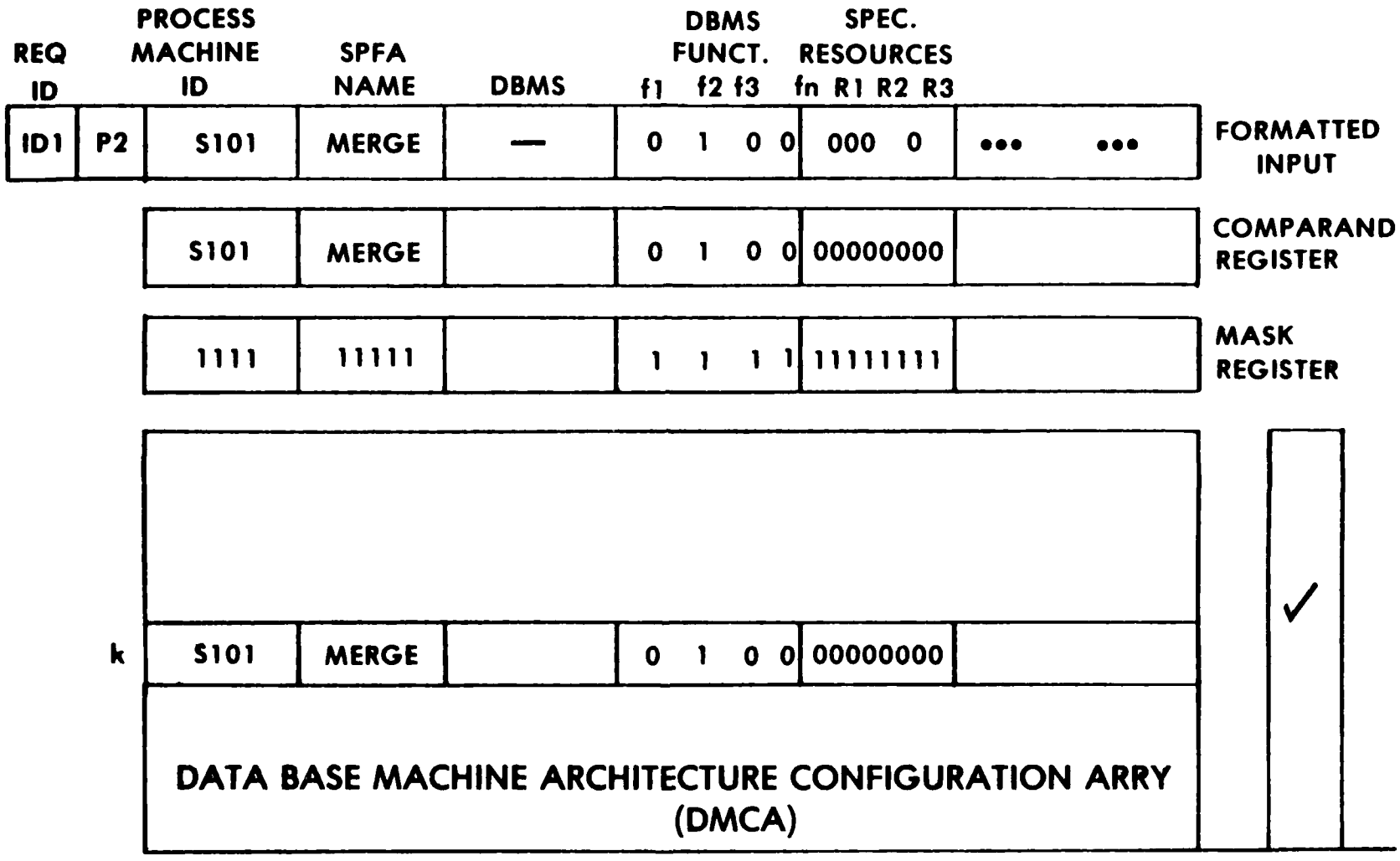
PROCESS: Create SPFA

NAME: "MERGE" SPFA

DBMS FUNCTION: f_2

This request is to create a "MERGE" SPFA where f_2 is the DBMS merge function. First, input to the CIM is formulated from the process description language. The formation of this input is illustrated in Figure 5-11 where a request identification is assigned and the create process is identified. S1 is assigned as the machine identification, the SPFA machine bit is set, and the f_2 DBMS function is identified. Once the formatted input is complete, a create process query is generated. This query performs the following:

(1) loads the CIM Comparand Register with the formatted input,



EXAMPLE CREATION OF A SPFA MACHINE

FIGURE 5-11

(2) identifies a SPFA machine attributes using the Mask Register, and

(3) OR's the contents of the Comparand Register into the first available DMCA entry.

Illustrated in Figure 5-11 is the SPFA machine created for this example. Entry k is the S1 SPFA machine. It is assigned the f_2 DBMS function.

Following creation of this machine entry, control returns to the SHM to develop a description. The Configuration Request Module (CRM) identifies the interface to the SHM programming environment to develop the SPFA description. Once completed the location of machine loadable code, in machine storage, is entered in the DMCA. An example of the process description language used to enter this location is shown below:

REQUEST ID: ID1
PROCESS: Complete Create
MACHINE ID: S1
LOCATION: @ MERG

The input defines the S1 machine and the location of its machine loadable code as at @ Merge. The machine identification and location are formatted for input to the DMCA. A complete create process query is generated and performs the following:

(1) loads the Comparand Register with the S1 SPFA machine,

(2) loads the Mask Register to identify SPFA machine,

(3) performs an equal to comparison to identify the S1 machine entry,

(4) sets the Mask Register to the SPFA location field, and

(5) OR's the location into the SPFA machine entry.

When completed, this S1 SPFA machine is available to be configured for execution. Test, evaluate and substitute processes for this machine can now be requested. The facility components needed to configure and control execution of a SPFA machine are the Master Configuration Control Machine (MCCM) and the Data Base Function Execution Machine (DBFEM). The utilization of the MCCM is now described.

5.6 MASTER CONFIGURATION CONTROL MACHINE

Described in this section are execution configuration requests that enter Master Configuration Control Machine (MCCM). These requests enter an Input Decoder Module (IDM). The IDM classifies the request type and gives control to the Configuration Control Module, as shown in Figure 5-12.

5.6.1 Configuration Control Module

This module consists of a:

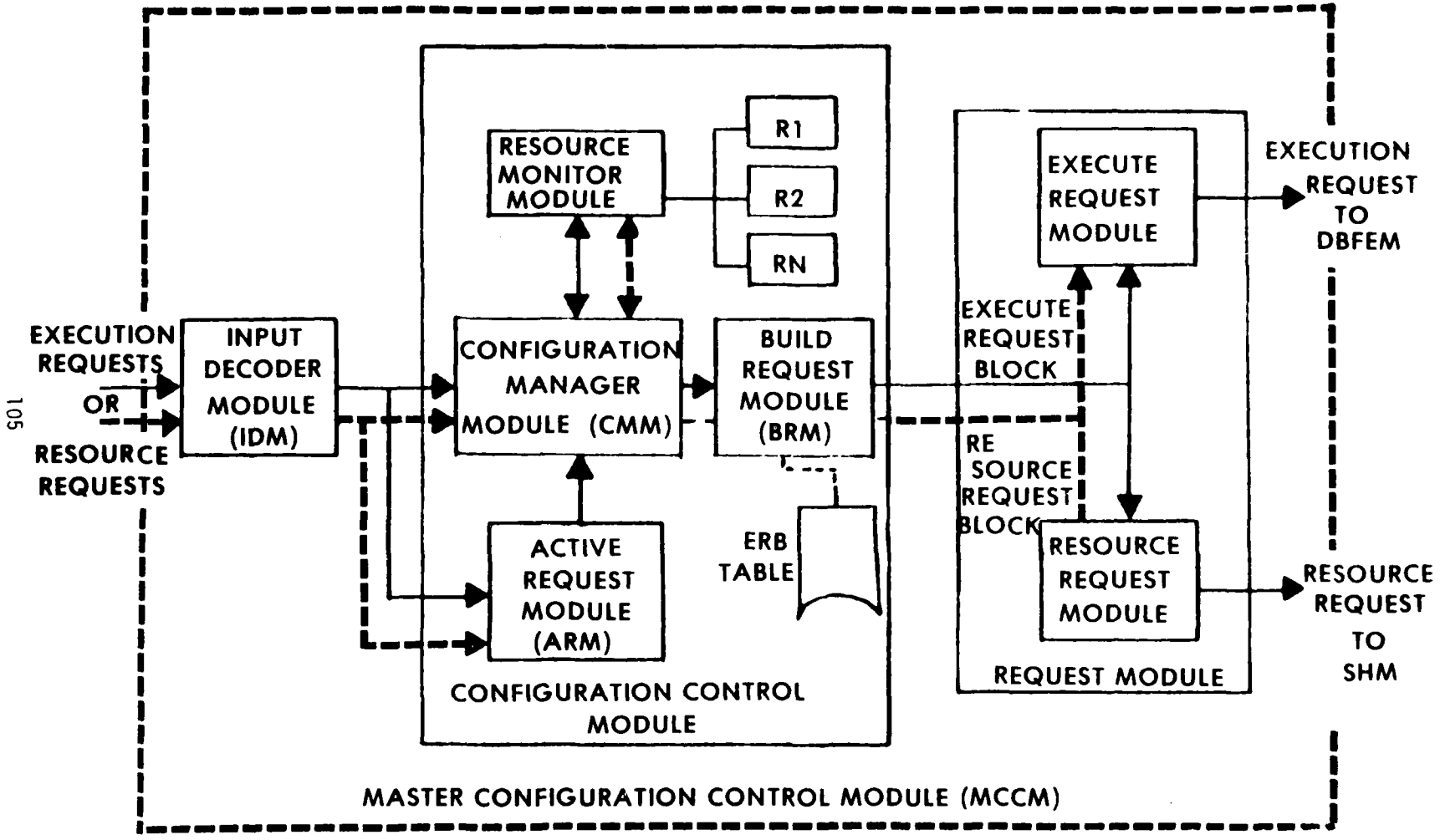
(1) Configuration Manager Module,

(2) Resource Monitor Module,

(3) Build Request Module, and

(4) Active Request Module.

Execution requests are passed to the Configuration Manager Module (CMM), while resource requests are given to the Active Request Module (ARM). The CCM determines availability of resources for an execution request. If resources are available, a Build Request Module (BRM) develops an Execution Request Block (ERB). Unavailability of resources causes the configuration request to be requeued. The format of this block, as a Request Block, is shown in Figure 5-13. First, the request



FLOW WITHIN THE MASTER CONFIGURATION CONTROL MACHINE

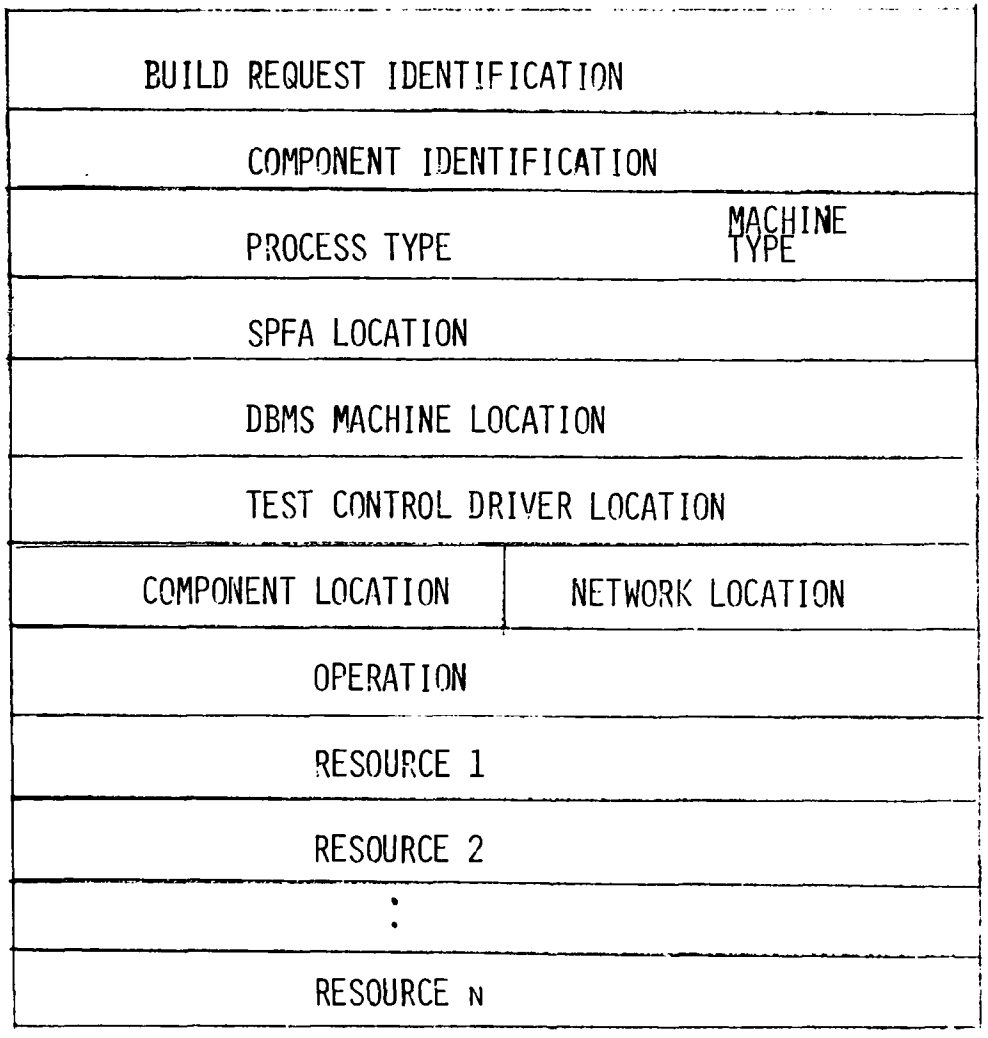
FIGURE 5-12

identification is defined. Next, the execution machine, DBFEM is identified, the process type is noted, and the machine type and its location are identified. Finally, the set of resources needed to configure this machine for execution are defined. When an ERB is complete it is included and marked active in an ERB Table (Figure 5-12). Each ERB is passed with an execution request to the DBFEM. Each ERB remains active in the ERB Table until the machine terminates execution on the DBFEM. Any resources requested by the machine during execution must be validated.

These requests enter the MCCM and are passed directly to the Active Request Module (Figure 5-12). This module determines if the resource request is valid. This validity is determined by comparing the machine request with the set of active ERB's. If valid, the ARM determines resource availability from the Configuration Manager Module. If the resource is available, the ARM passes control to the Build Request Module. This module develops a Resource Request Block (RRB). The format of this block is also based on the Request Block illustrated by Figure 5-13. However, in this case, the component is the SHM. The resource needed and the desired resource operation are identified. When the RRB is complete, the Request Module (Figure 5-12) issues a special resource request to the SHM.

5.6.2 Request Module

The Request Module is composed of an Execution Request Module and a Resource Request Module. The Execution Request Module uses the Execution Request Block to generate an execution request to the DBFEM. The Resource Request Module uses the Resource Request Block to generate a resource



SAMPLE FORMAT OF A REQUEST BLOCK

FIGURE 5-13

request to the Service Host Machine. The flow of resource requests from the MCCM to the SHM were described in section 5.2.2. The execution requests to the DBFEM are described in the following section.

5.7 DATA BASE FUNCTION EXECUTION MACHINE

The Data Base Function Execution Machine is configured as a specific machine type for each execution request. The actual configuration is performed when an execution request is received. These requests enter an Input Control Module.

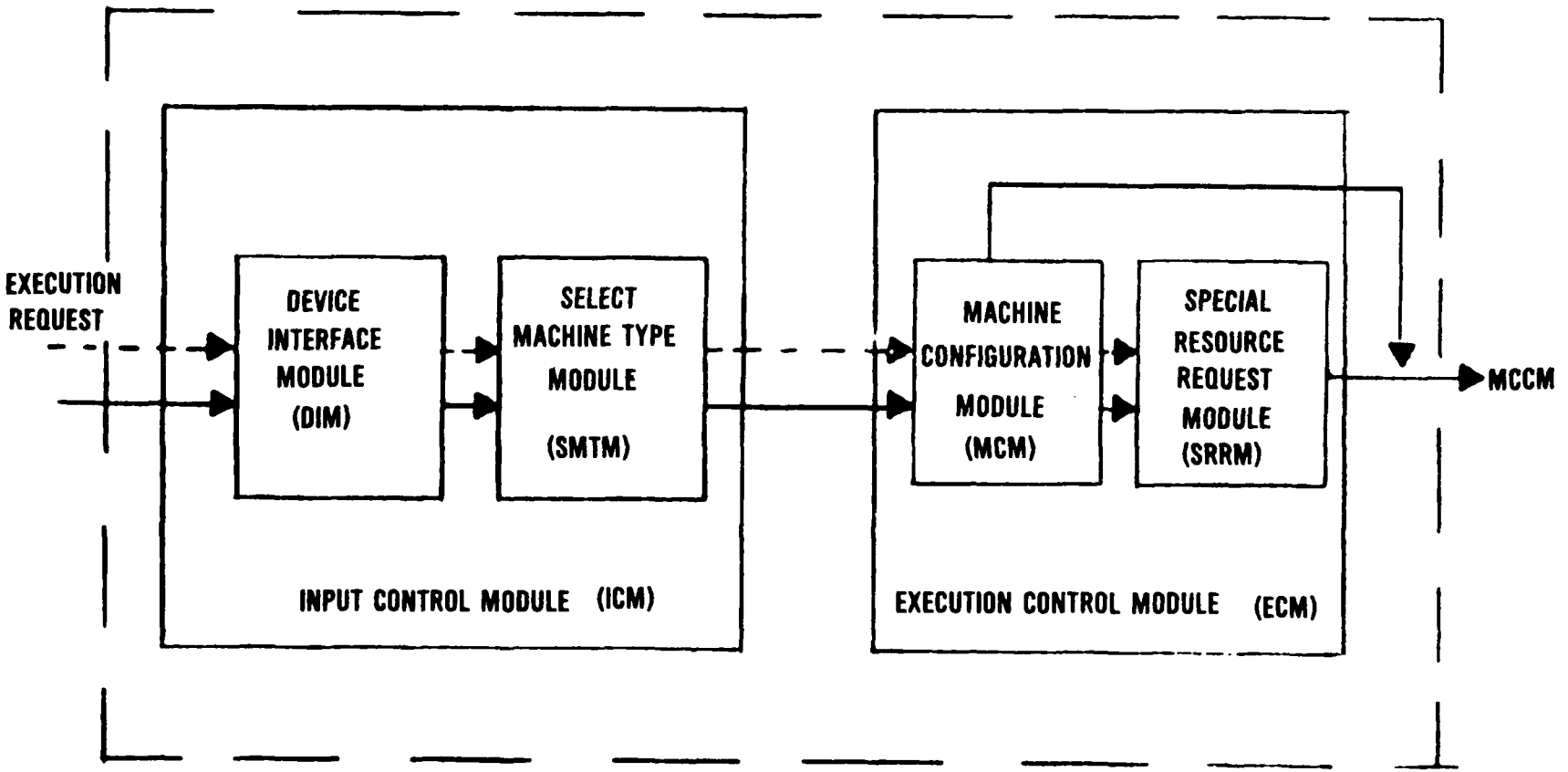
5.7.1 Input Control Module

The DBFEM Input Control Module, illustrated in Figure 5-14, consists of:

(1) Device Interface Module (DIM), and

(2) Select Machine Type Module (SMTM).

The Device Interface Module interfaces the DBFEM to the MCCM. Each execution request passes through this interface to the Select Machine Type Module. This module examines the request and classifies it as a new execution request or a resource completion request. For a new request, the DBFEM is configured as a new machine. For a resource completion request a previous machine is restored. The type of request is classified and passed to the Execution Control Module (ECM).



FLOW WITHIN DATA BASE FUNCTION EXECUTION MACHINE

FIGURE 5-14

5.7.2 Execution Control Module

The ECM (Figure 5-14) consists of a:

(1) Machine Configuration Module, and

(2) Special Resource Request Module.

The Machine Configuration Module (MCM) uses the input Execution Request Block to configure the execution machine type. Three types of execution machines, illustrated in Figure 5-15, are possible.

(a) SPFA Machine,

(b) DBMS Machine, or

(c) DBMS/SPFA Machine.

A DBMS machine is used for the select candidate process. A SPFA machine is used for test and evaluate processes and a DBMS/SPFA Machine is used for substitute processes. For each machine type either a SPFA, a DBMS or a DBMS/SPFA Control Module is configured. Several control modules may be available in the facility and can be specified as part of the input process description language. The control module requires different sets of input for each machine type.

For a DBMS machine the control module requires:

(1) location of DBMS machine description,

(2) location of DBMS and system support software, and

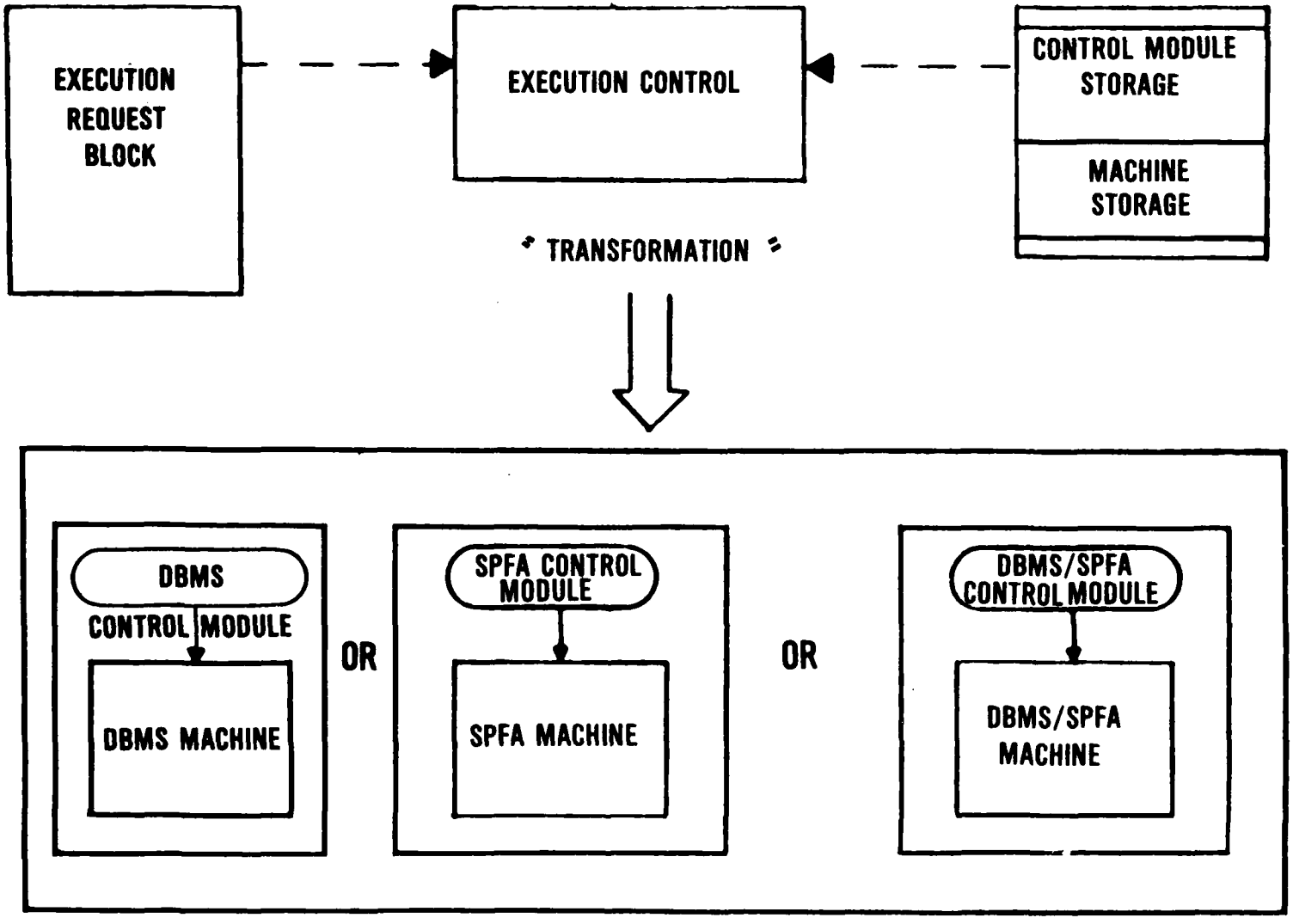
(3) location of application programs.

For a SPFA machine the control module requires:

(1) location of SPFA machine description,

(2) realization timing control array assignments,and

(3) machine driver location.



111

CONFIGURATION OF A MACHINE TYPE ON THE DBFEM

FIGURE 5-15

The DBMS/SPFA Control Module requires:

(1) location of SPFA Machine description,

(2) location of DBMS Machine description,

(3) location of a data base management system,

(4) location of a SPFA Input Generator (SPFA/IG), and

(5) location of a software application program.

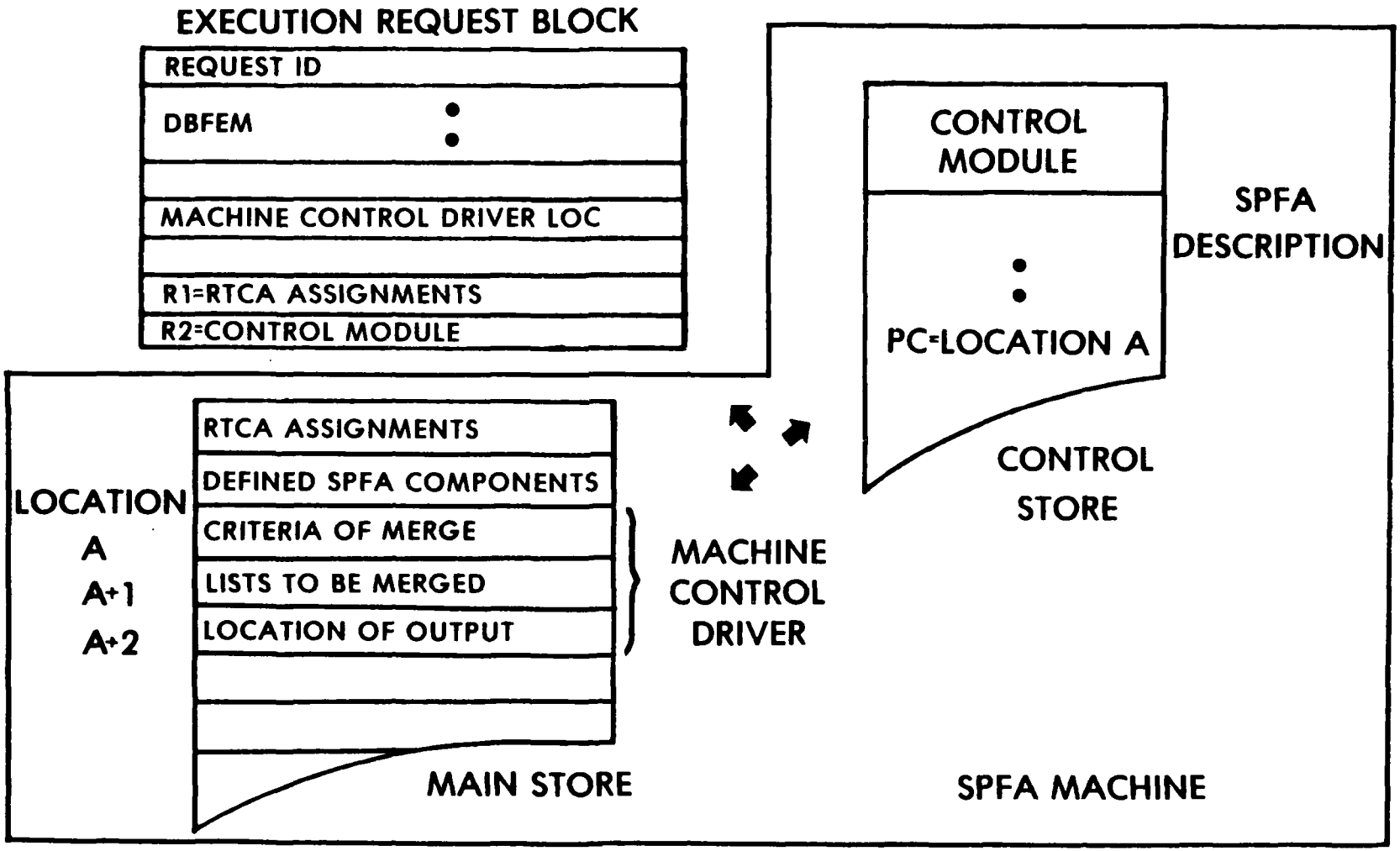
In addition, the DBMS/SPFA Control Module must support concurrent emulations of two or more machines under control of a virtual data base machine monitor. Once the control module configures a machine type, the machine is available for execution. During execution the test, evaluate and substitute processes using the machine can be performed. These processes are now described.

5.8 PROCEDURES FOR TEST PROCESS FUNCTIONS

The testing process is available to insure that a SPFA machine executes properly and performs its intended DBMS function. This process is conducted following configuration of the SPFA machine.

5.8.1 Execution of SPFA Machine

Each SPFA machine is configured with a SPFA description and machine control driver. This machine begins execution, as illustrated in Figure 5-16, with input from its machine control driver. The input from this driver dictates the sequence of operations executed by the SPFA machine. For example, if the SPFA machine performs a "MERGE" DBMS function, a machine driver control may consist of the following arguments:



EXECUTION OF A SPFA MACHINE

FIGURE 5-16

(1) type of "MERGE", i.e. OR, AND, NOT,

(2) lists to be merged, and

(3) location of output list.

The SPFA machine then executes a sequence of operations, consistent with its architecture description, needed to complete the "MERGE" DBMS function. It is this sequence that needs to be functionally verified by the data base machine architect. The verification can be performed as follows.

5.8.2 SPFA Machine Functional Verification

A verification procedure is performed by examining the state of the SPFA machine at selected breakpoints. As the SPFA machine executes, and a selected point is reached, execution is halted. In this state, the data base machine architect has a set of tools, that permit access to all portions of the SPFA machine.

A procecure for this verification is illustrated in Figure 5-17. A set of commands are available to examine portions of the SPFA machine:

(1) display main store location,

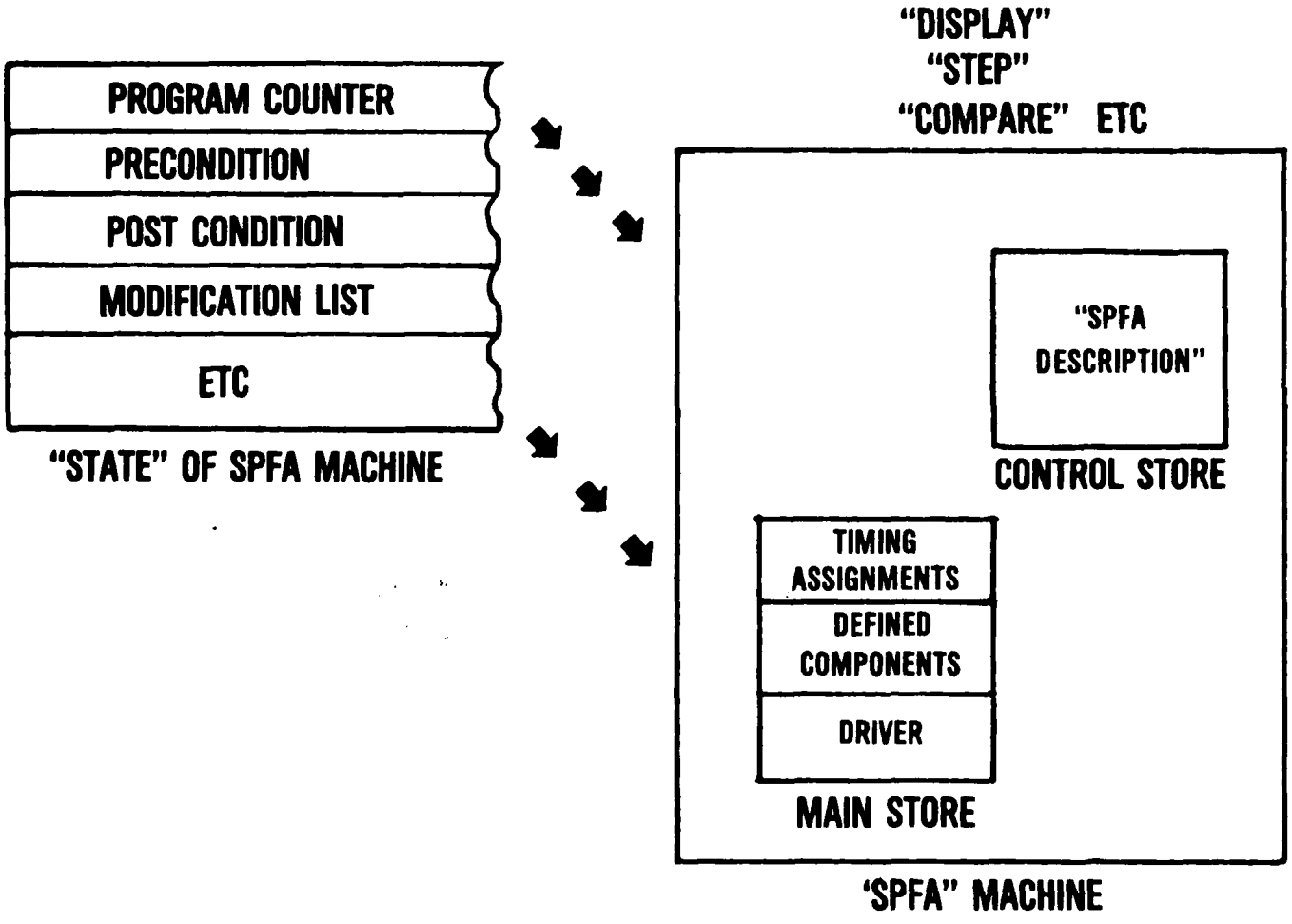
(2) display control store location,

(3) step SPFA execution,

(4) compare location,

(5) notify change in location,

(6) etc.



A VERIFICATION PROCEDURE FOR A SPFA MACHINE

FIGURE 5-17

The type of commands can vary according to the Control Module. The data base machine architect can visually inspect all parts of the SPFA machine using these commands. One procedure for verification can use Crocker's(CROC78) state delta technique. Using this technique, a modification list can be constructed for a set of data items changed as a SPFA machine executes. Figure 5-17 illustrates these values being displayed during a selected state of the SPFA machine.

These values are examined to identify changes in states between a pre-condition and a post-condition. If values in these states are changed, as a result of moving between states, with the exception of Crocker's modification list, this can signal an invalid state. During each state change, the pre-condition and post-condition lists are examined by the data base machine architect. If the data base machine architect detects an invalid post condition list, then the SPFA machine is declared not verified. Once a SPFA machine is declared invalid, debugging procedures can be instituted.

5.8.3 SPFA Machine Debugging

A SPFA machine debugging procedure uses the Debugging Process Model developed by Finfer (FINF79). The model defines the following activities for debugging:

(1) verification,

(2) localization,

(3) execution analysis,

(4) hypothesis formulation, and

(5) resolution.

The first activity, verification, substantiates that a SPFA machine state is invalid. Next, the error that causes an invalid SPFA machine must be localized. The localization of the error is performed by establishing a set of bounds. This can be accomplished by stepping the execution of the SPFA machine at the micro-instruction level. During each step, the SPFA machine state is examined. Each step is controlled by selecting the number of micro-instructions between steps. Next, the number of micro-instructions allowed to execute between steps is reduced until the error occurrence has been sufficiently bounded. Once the error source has been bounded, an execution analysis is performed.

The execution analysis examines the SPFA machine, in both main and control store, including its registers and other machine identifiable features in specific states. Values are established for the machine in each state. A set of assertions are developed for specific variables for each of these states. These assertations are re-examined as the SPFA machine steps through execution until

found invalid. Once found invalid, a hypothesis formulation is conducted. The hypothesis formulation is used to analyze if the SPFA machine state in error violates a set of predefined assertions. If an error is identified at microprogram instruction level, it is traced back to the hardware description language source level program that is used to generate the instruction. Next, the source hardware description language is examined. A resolution activity determines if final cause of the error is within the source language. If the SPFA is described inaccurately, it is redescribed. A create process is used to describe a new SPFA. This SPFA replaces the previous SPFA that was entered in the facility. If the error is determined to be machine oriented, the machine can be reconfigured and re-executed.

5.8.4 Test Process Example

The following example illustrates a process request to test a SPFA machine. The process description language for this request is shown below:

```
PROCESS :   TEST
MACHINE ID :   S1 SPFA
NAME :    MERG1
CONTROL MODULE :  EC1
MODE :   INTERACTIVE
DRIVER :   @ Testdrive
```

This example requests testing of the MERG1 version of the S1 SPFA machine. Interactive mode is requested. The EC1 Control Module is also requested. Finally, the location of a machine driver

"Testdrive" is defined. Illustrated in Figure 5-18 is the formatted input for this example. In this example, the machine ID and name are used to identify the S1 SPFA. A test process query, for this example, has been generated. A sample portion of this query is illustrated in Figure 5-19 using the instruction set of an associative unit developed by Singhania (SING77). This query is loaded and executed on the CIM. The query directs the CIM's Comparand Register to be loaded. A search of the DMCA is made for the S1 "MERG1" SPFA machine (Figure 5-13). If an S1 machine is not found, the user is notified that this machine is not available in the DMAD Facility. In this example, the S1 machine is identified, as illustrated in Figure 5-18, at entry k. The machine storage location for the S1 SPFA is at location l_1. In addition, this machine requires the r2 resource.

All resources identified are returned to the SHM and the Configuration Request Module. The CRM includes these resources to develop a Configuration Request Block (CRM). An example of this block, illustrated in Figure 5-20, includes the following:

(1) type of process,

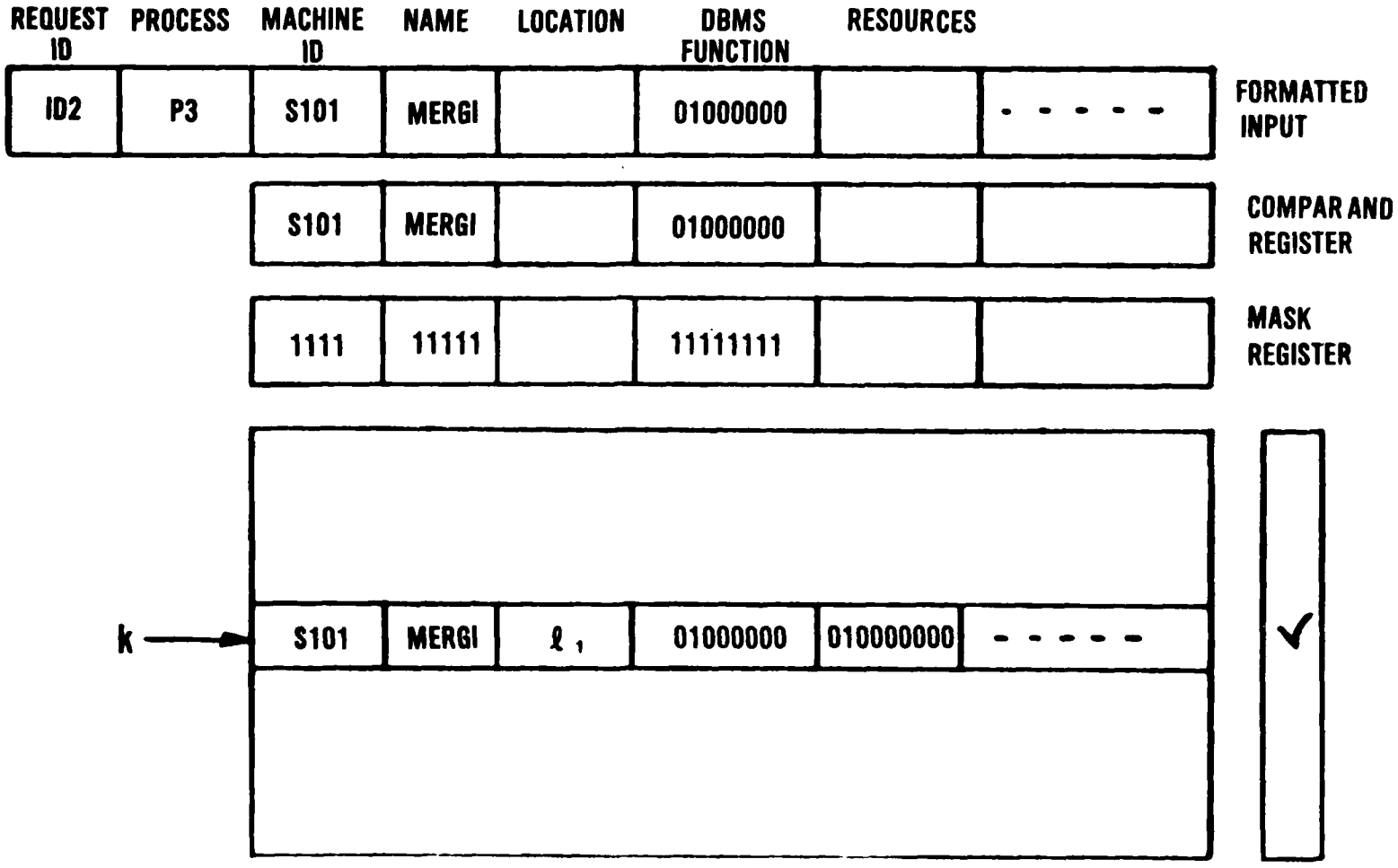
(2) request type and component,

(3) SPFA loadable location from machine storage, l_1,

(4) Special Resources, r2,

(5) Control Module - EC1, and

(6) machine driver location, "Testdrive".



EXAMPLE SPFA MACHINE IDENTIFICATION

FIGURE 5-18

```
LMR     SPFA    -   LOAD SPFA
                    MASK - FIELDS 1,3,4

LCR     RR      -   MOVE REQUEST REGISTER
                    TO CR

EQT             -   EQUAL TO SEARCH

MRS     T       -   RESPONSES IN T

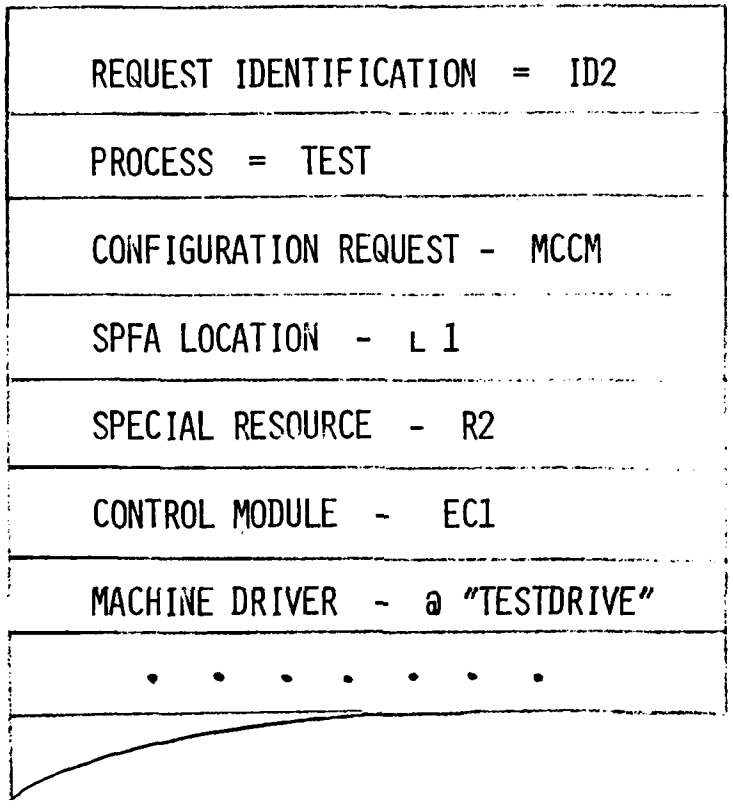
                -   RETURN IF EMPTY

LMR     R       -   IDENTIFY RESOURCES AND
                    SPFA LOCATION

MRS     B       -   MOVE RESPONSES TO OUTPUT BUFFER
```

SAMPLE TEST PROCESS QUERY

FIGURE 5-19

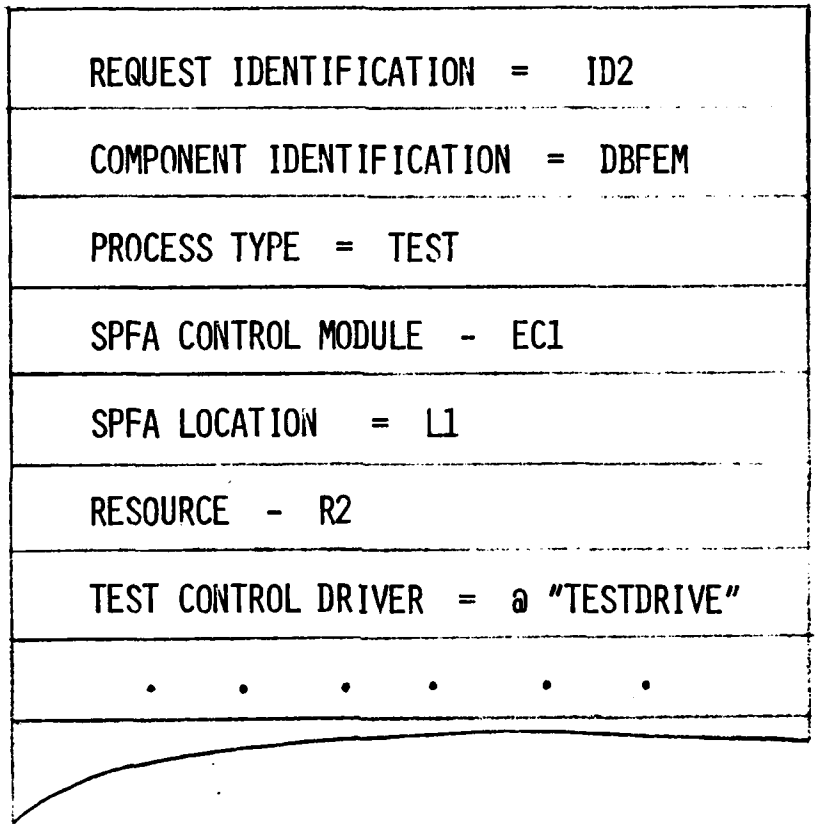


CONFIGURATION REQUEST BLOCK

FIGURE 5-20

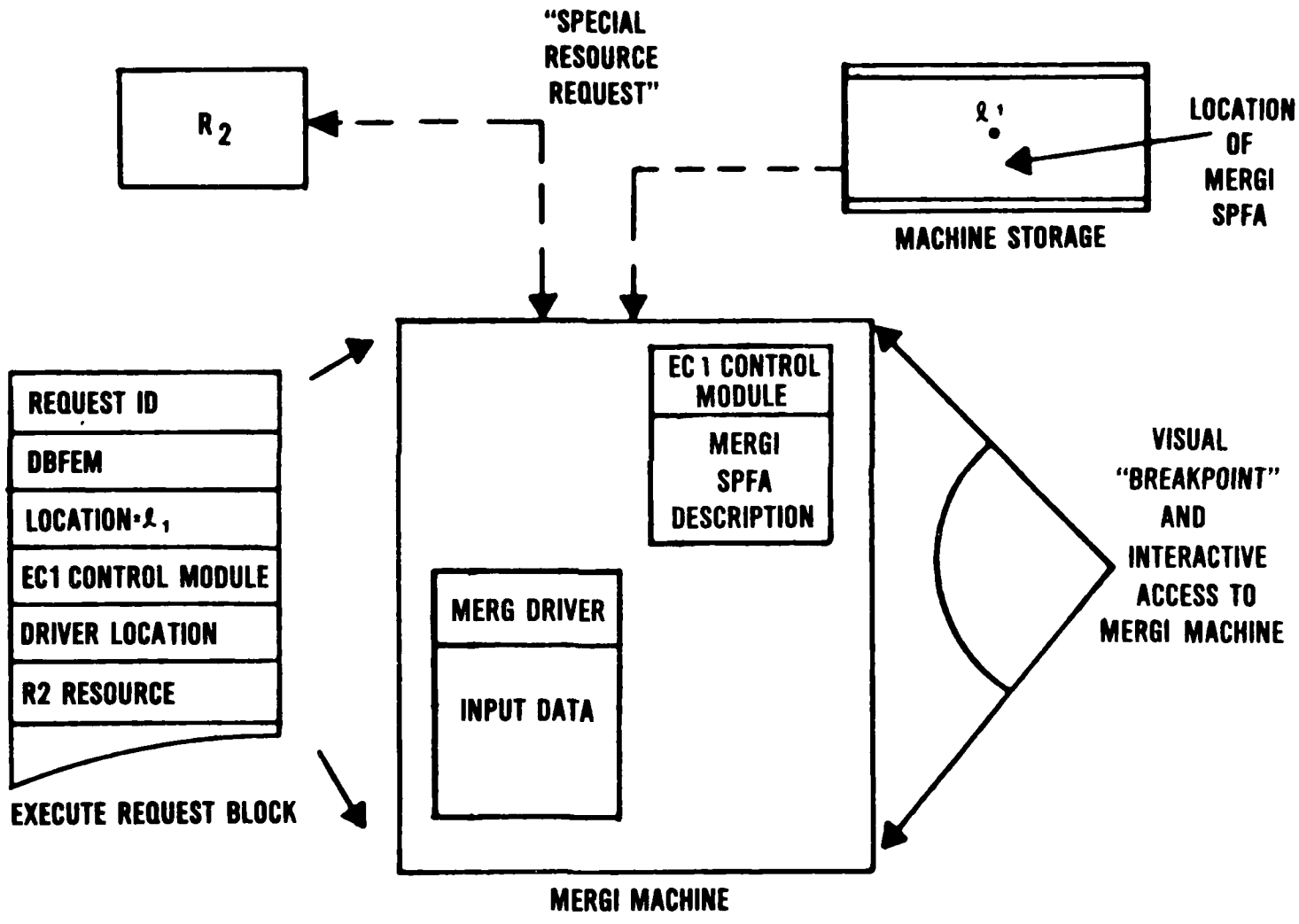
The CRB, along with an execution configuration request, is passed to the MCCM, For this request, the r2 resources must be available. When the MCCM confirms r2 resource availability, it builds an Execution Request Block (ERB). An example ERB for this example is illustrated in Figure 5-21. This ERB identifies the location of the SPFA machine, location of DBFEM, the SPFA Control Module, and the SPFA machine driver location. This ERB is passed with the execution request to the DBFEM. The Input Control Module on the DBFEM confirms this as a new machine request and initiates the Execution Control Module. The ECM initializes the EC1 Control Module. Next, the EC1 Control Module configures the MERG1 SPFA machine, as illustrated in Figure 5-22.

The MERG1 "S1" SPFA description is loaded in control store. The "Testdrive" driver is initialized in main store. Interactive mode is provided by the EC1 Control Module and is used to examine the state of the SPFA machine during execution. The r2 resource is treated as a special resource that can be called from the SPFA machine during execution. Once execution begins, the execution path of this machine is followed,in detail, to verify its capability to perform the f2 DBMS function. A data base machine architect continues this verification until he/she is satisfied the SPFA machine executes correctly. Once a SPFA machine has been tested, timing assignments from the realization timing control array can be added to its configuration. The addition of these timing assignments are part of the evaluation process. This process is described in the next section.



EXECUTION REQUEST BLOCK

FIGURE 5-21



CONFIGURATION OF SPFA MACHINE FOR A TEST PROCESS

FIGURE 5-22

5.9 PROCEDURES FOR EVALUATION PROCESS FUNCTIONS

The evaluation process enables a user to utilize the facility to evaluate and compare alternative SPFA's that may perform the same DBMS function. With this process a procedure is used to assign timings to operations groups that comprise parts of a SPFA description. These timings are then entered into the SPFA's realization timing control array. The location of the RTCA, for each SPFA, is defined as part of a DMCA entry. The times that are entered in the RTCA are determined using an identification of hardware operations function. This function is described below.

5.9.1 Identification of Hardware Operations

The identification of hardware operations function is required for an evaluation process. This function identifies a set of operations groups from each SPFA description. These operations are entered in column 1 of the RTCA. These operations are specified at a functional level in the description language of a SPFA machine. These are the operations that can eventually be performed in a hardware version of the SPFA by an integrated circuit, Thus, times for the operations are based on an assumed set of hardware characteristics. For instance, the following statement from a SPFA description:

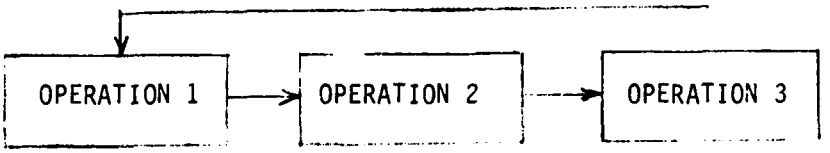
Statement n : R 1 ⟵ R 2

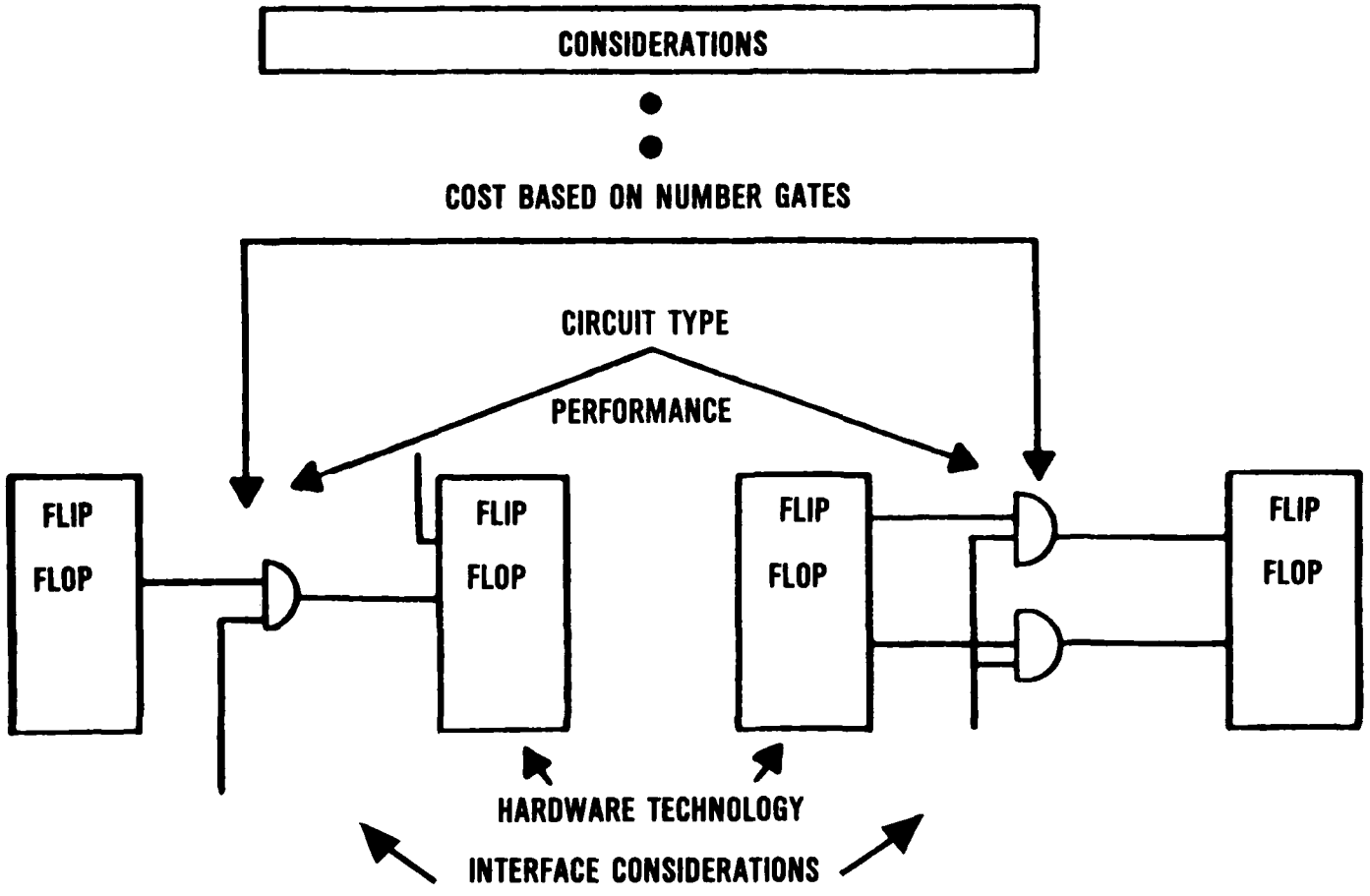
is a typical transfer of information transfer operation specified in a hardware description language. This statement specifies the transfer

of the contents of register R2 to register R1.

An analysis of this operation in Figure 5-23 reveals several considerations can affect its performance. First, the type of transfer can be considered. For example, two types of information transfer circuits have been described by Foster (FOST70). For these circuit types, a jam circuit is twice as fast as a clear and copy but requires more gates. Thus, a cost versus performance tradeoff may be analyzed for this operation. Similarly, other types of tradeoffs may be examined. Some of these tradeoffs include variations in performance, cost, reliability for various hardware technologies, as illustrated in Table 5-1. These values illustrate a range in variability depending on the hardware technology chosen. This table not only illustrates gate delay of the technology but other factors that may vary including its reliability, cost, heat, etc. Each of these factors may be an important consideration choosing and tailoring the SPFA machine for a specific application.

In addition to this type of analysis the facility can also be utilized to examine architectural features of a SPFA. For instance, a SPFA may have several alternatives for describing parallelism of its operations. As illustrated below, one description of a set of operations may be sequential.





TYPES OF PERFORMANCE CONSIDERATIONS FOR SPFA OPERATIONS

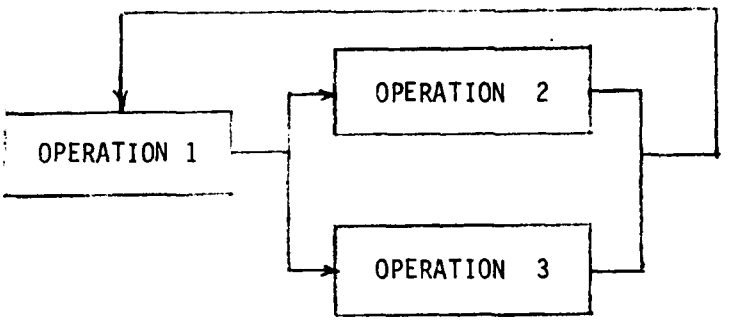
FIGURE 5-23

TECHNOLOGY	TYPICAL DELAY/GATE	TYPICAL POWER/GATE	TYPICAL COST/GATE
PMOS	100 ns	0,2 mw	0.1/-2¢
NMOS	50 ns	0.2 mw	0.1/-2¢
CMOS	25 ns	10.0 uw	10-30¢
SHOTTKEY TTL	3 ns	20. mw	25¢
LOW-POWER SHOTTKEY	10 ns	2. ms	25¢
ECL	2 ns	30. mw	30-40¢

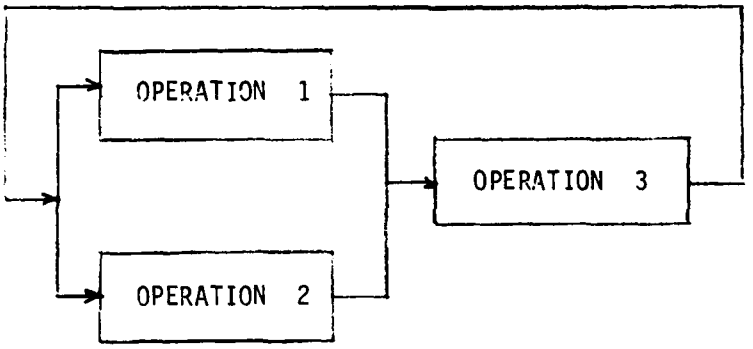
CHARACTERISTICS OF SEMI-CONDUCTOR TECHNOLOGIES

TABLE 5-1

However, the facility can be utilized to examine alternative ways to describe this architecture. In one case it may be described as:



or it may be described as:



The evaluation process should consider these factors and devise a strategy that optimizes the choice of SPFA for a specific application. Thus, this can be done using the facility to compare alternative SPFA's that are based on different architectural constructs. Their performance is obtained by executing each SPFA machine under the same set of conditions and collecting timings using the facility.

In order to illustrate how these times can be generated, a realization timing control array (RTCA) is shown in Figure 5-24. Column one represents a set of operations that have been identified in each SPFA description. Columns 2 to n+1 classify times for each operation based

CHARACTERISTICS / SPFA OPERATIONS	REALIZATION ASSUMPTION (RA1)	REALIZATION ASSUMPTION (RA2)	. . . . .	REALIZATION ASSUMPTION (RA n)
	TIME, COST, RELIABILITY ECT			
OPERATION 1	t(1,1) c(1,1)			
OPERATION 2 • •				
OPERATION n	t(n,1) c(n,1)			t(n,m) c)n,m)

ENTRIES IN A REALIZATION TIMING CONTROL ARRAY (RTCA)

FIGURE 5-24

on the characteristics of a realization assumption (RA). For instance, realization assumption 1 can assume a strategy that uses high speed logic at high costs for the operation. Realization assumption 2 may propose lower speed logic for the same operation at lower cost. These changes in realization assumption may affect time of an operation compared on this basis.

The actual assignments for the RTCA are made by choosing the time from an entry as t(m,n) which is the time to perform the operation m using realization assumption n. Other entries may be added similarly such as cost as c(m,n). Using the realization timing control array a performance determination can then be made.

5.9.2 SPFA Machine Performance

A complete assignment of timing values from an RTCA for each SPFA machine is made prior to an execution request. An example of how the timing from a RTCA can be used to determine performance is illustrated in Figure 5-25. A set of six operations 1 - 6 from a SPFA description are identified in the RTCA. The operations for statement numbers 1 and 6 are non-executable and are not assigned a timing. Statement numbers 2,3,4 and 5 require variations of time (t,t1,t2 and t3) according to the realization assumption.

An input assignment may choose the following:

Operation	Execution One	Execution Two
1	RA (1,1)	RA (1,2)
2	RA (2,1)	RA (2,2)
3	RA (3,1)	RA (3,3)
4	RA (4,1)	RA (4,1)
5	RA (5,1)	RA (5,1)
6	RA (6,1)	RA (6,2)

SPFA OPERATION	REALIZATION ASSUMPTION (1)	REALIZATION ASSUMPTION (2)	REALIZATION ASSUMPTION (3)
1	–	–	–
2	t	t+1	t+2
3	t1	t1+1	t1+2
4	t2	t2+1	t2+2
5	t3	t2+1	t3+2
6	–	–	–

REALIZATION TIMING CONTROL ARRAY

SPFA DESCRIPTION	STATEMENT NO.
:	
BEGIN	1
IF X-REG;	2
OPERATION GROUP 1	3
IF Y-REG;	4
OPERATION GROUP 2	5
END	6

EXAMPLE REALIZATION TIMING CONTROL ARRAY UTILIZATION

FIGURE 5-25

RA (1,1) identifies for operation 1 the time determined by realization assumption (1). This time is 0. RA (2,1) identifies for operation two the time determined by realization assumption (1). This time is t. The remaining entries are chosen similarly. The set of times are used as part of the SPFA machine configuration for execution one.

Next, assume that the SPFA begins execution one. The set of operations required is shown in Figure 5-25. Whenever operation 1 is required, the timing control clock is updated by its assigned value from the RTCA. Execution one continues until all operations required have been completed. At this point, the accumulated timing data is collected for execution one. If another execution is requested, the SPFA machine is re-configured with a new set of RTCA timing assignments. Execution two begins with a different set of timings for the same SPFA execution sequence. The accumulated timing data is also collected for this execution.

In this example, if execution one constitutes the execution of statements 1 to 6, then the accumulation of time is t + t1 + t2 + t3. For execution two, the accumulation of (t + 1) + (t1 + 2) + t2 + t3. These results are collected and can be later analyzed by the data base machine architect following completion of all executions.

5.9.3 Evaluation Process Example

Assume that two SPFA's have been developed to perform the Merge DBMS function. Each SPFA is based on different architectural considerations. One procedure to evaluate these SPFA's is to use the facility to generate timings for each SPFA machine. An illustration of a request to determine performance of the first "MERG1" machine is illustrated by

the process description language below:

PROCESS DESCRIPTION; Evaluation

SPFA NAME: MERG1

CONTROL MODULE: EC1

MACHINE DRIVER: @ Testdrive

EXECUTION: One

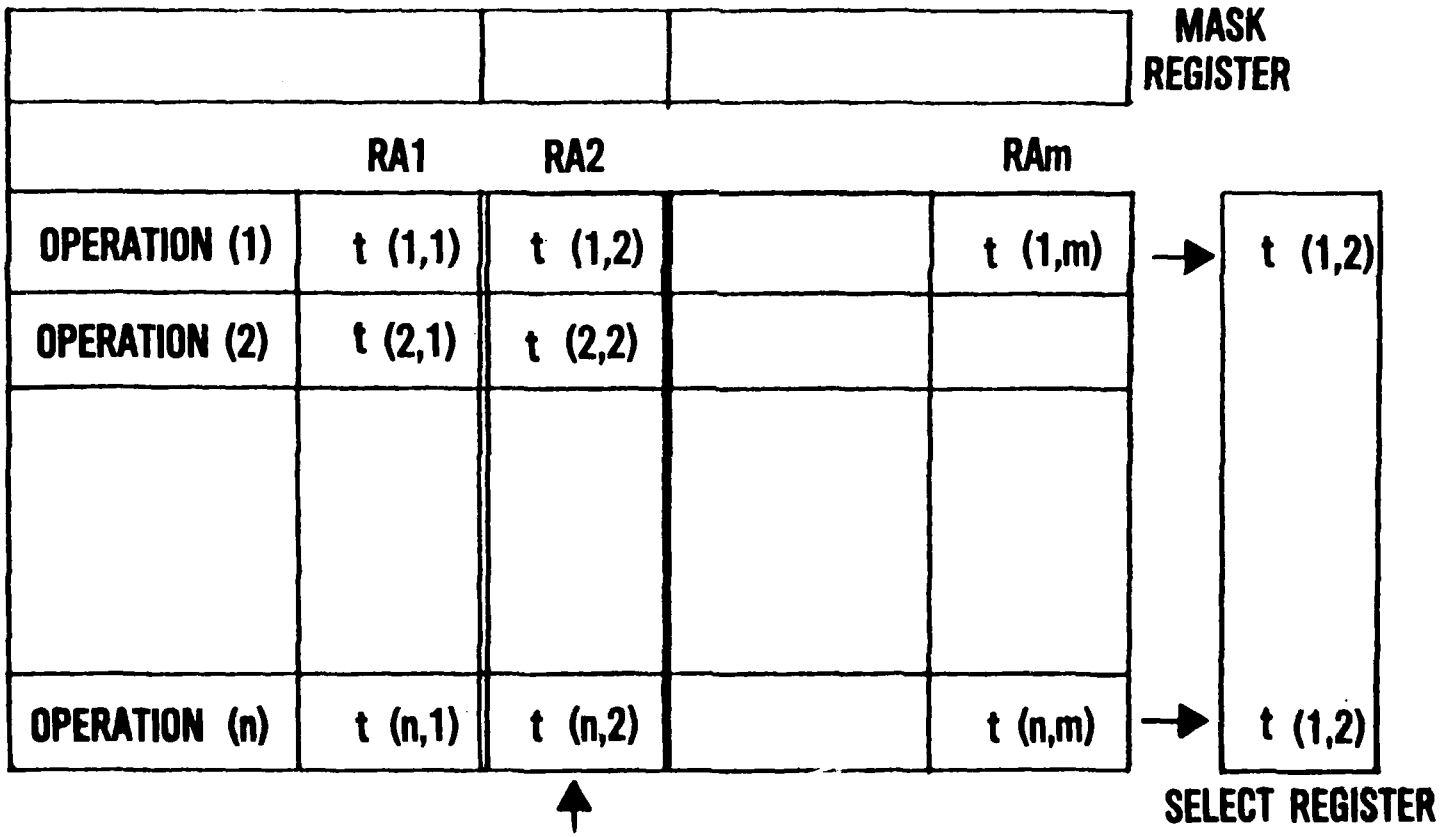
REALIZATION TABLE: A, RA2

This requests an evaluation of the "MERG1" SPFA. It is assumed that its RTCA has been defined and generated. The RTCA assignment for execution one is defined as (A,RA2). This can be interpreted as all assignments from realization assumption two. A bit slice from the RTCA RA (1,2) through RA(n,2), as illustrated in Figure 5-26, can provide these assignments. These assignments are moved to the select register. The assignments in this register are moved to the output buffer and returned to the SHM.

Next, assume the facility is configured as the "MERG1" SPFA. The "MERG1" SPFA description is composed of a set of operations in DBFEM control store, as shown in Figure 5-27. The sequence these operations are executed depends on the input criteria of its driver "Testdrive". The RTCA assignments have been configured as part of the "MERG1" machine, as shown in Figure 5-27.

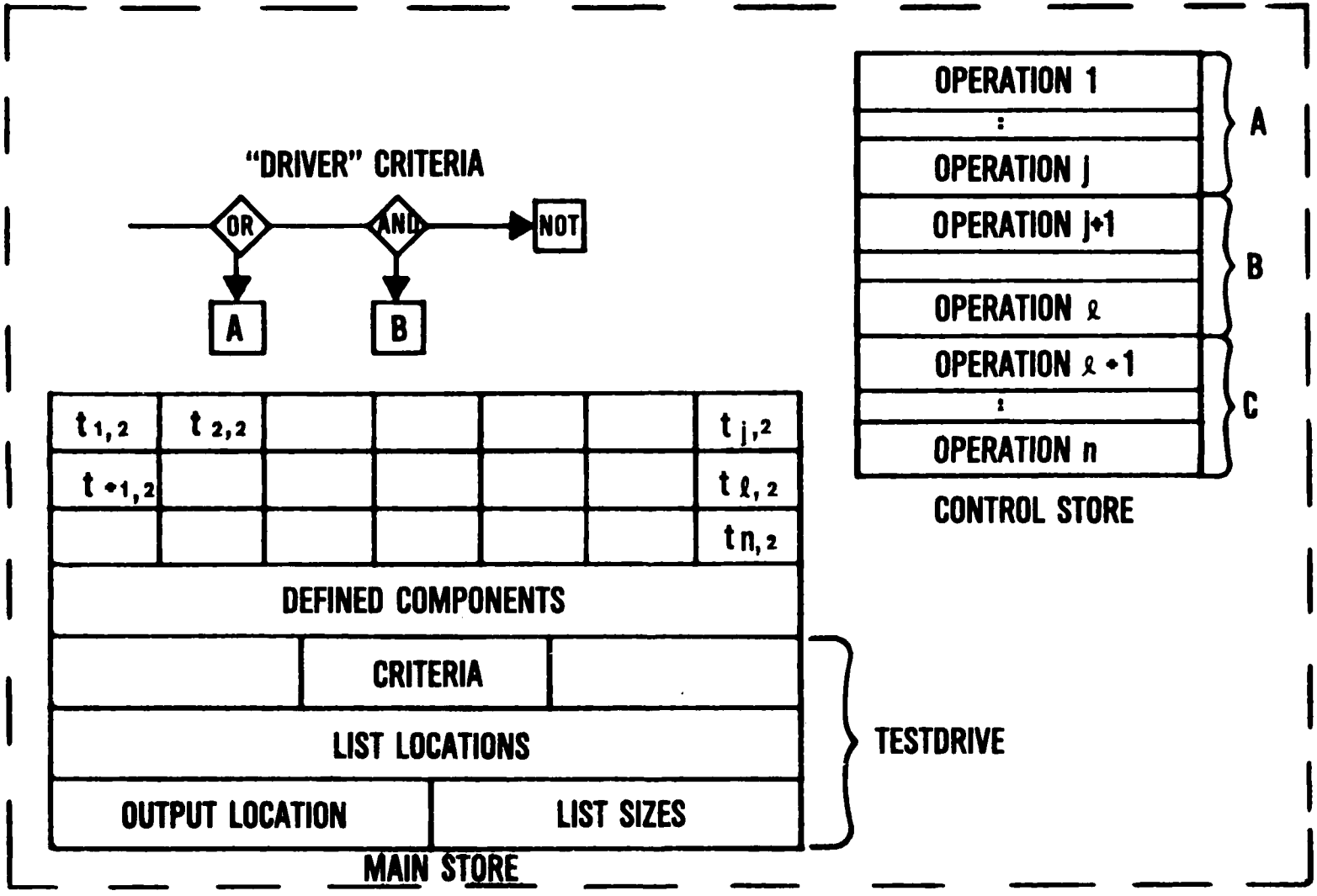
The SPFA machine begins execution by examining the contents of its driver. In this example, "Testdrive" includes three arguments illustrated in Figure 5-27. They include:

(1) a Merge criteria,



GENERATION OF TIMING ASSEGNMENTS USING A REALIZATION TIMING CONTROL ARRAY

FIGURE 5-26



CONFIGURATION OF SPFA MACHINE FOR EVALUATION PROCESS

FIGURE 5-27

(2) list location is identified, and

(3) an output location and list size.

Assume for this example that three sets of input criteria are possible for this DBMS function (OR, NOT, AND NOT). The input criteria determines the execution path. If, for example, the OR criteria is specified, then the set of A operations represent Operation 1 to Operation j. The values RA (1,2) to RA (j,2) are assigned to the group of A operations. When these operations execute, the timing control clock is updated by the amount assigned in t (1,2) through t (j,2). The total time accumulated when this operation completes is the performance of the "MERG1" SPFA for realization assumption 2. A similar timing is computed for B and C operations. Similar timings can be computed using different combinations of realization assumptions and input criteria. The evaluation process can continue for several iterations to highlight specific performance sensitivities.

Next, the same procedure is performed for the second merge machine. Using this type of analysis the facility can be used to help evaluate and analyze the individual architectural constructs of each SPFA. A complete analysis of each SPFA is made using this procedure to help determine if one should be selected as a SPFA for the given application. In addition, other SPFA's can be evaluated using this procedure, SPFA's may be modified and re-created, or combinations of SPFA's may be developed.

5.10 PROCEDURE FOR SUBSTITUTION PROCESS

The final SPFA development process is substitution. The DMAD Facility is configured as a DBMS/SPFA machine for this process.

5.10.1 DBMS/SPFA Identification

The DBMS/SPFA identification function identifies the configuration resources needed to transform the DMAD Facility into a DBMS/SPFA machine. The following procedures are used for this function:

(1) DBMS/SPFA machine identification, and

(2) DBMS/SPFA machine pre-processing.

The Data Base Machine Architecture Configuration Array (DMCA) is used for the DBMS/SPFA machine identification. Each machine is separately identified on the process query for a substitute process request. Once a SPFA machine and a DBMS machine have been identified, they are combined into a DBMS/SPFA machine where the SPFA machine entry replaces the DBMS machine entry for the common DBMS function. This common DBMS function is called the referenced DBMS function. The set of special resources needed to configure a DBMS/SPFA machine consists of the union of the resources required by the DBMS machine with the resources required by the SPFA machine.

Once a DBMS/SPFA machine is identified, a pre-processing function is required. This function prepares the DBMS/SPFA machine for substitution. The interfaces to the software DBMS function being replaced are now used as interfaces to the SPFA machine. These interfaces define the set of input and outputs required by the SPFA machine each time it is called. These calls may be made from the DBMS itself or from one of the software application programs used in the select

candidate process. Each call is identified and replaced with a special instruction.

The function trap instruction identifies that a SPFA machine is available to perform the DBMS function. The following procedures are performed for the DBMS/SPFA pre-processing:

(1) development of a SPFA input generator,

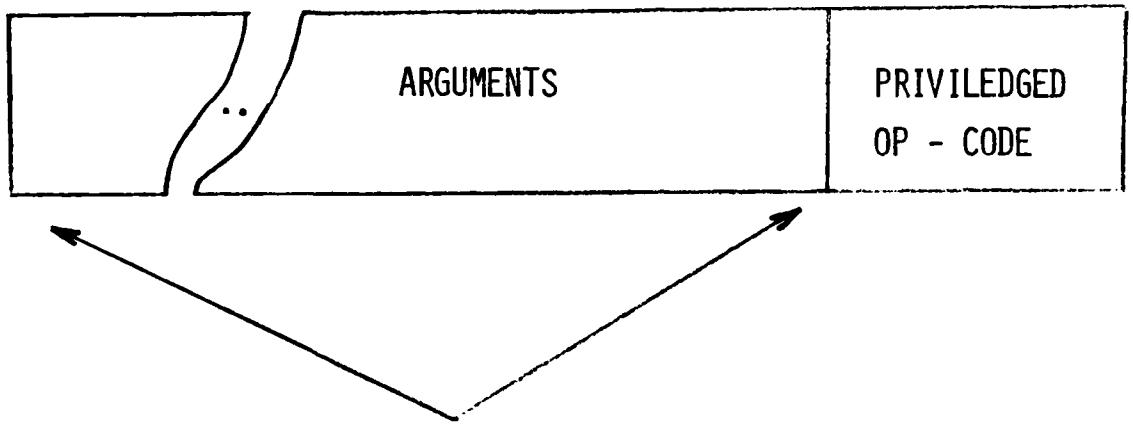
(2) development of a function trap instruction, and

(3) modification of referenced function call.

The SPFA input generator (SPFA/IG) is developed by the data base machine architect. It uses the input arguments from each referenced DBMS function call to generate the machine driver for the SPFA machine. A SPFA/IG can be developed to execute- directly on the DBFEM or called as a special processing module from the SHM.

A function trap instruction is especially created for each DBMS machine. This instruction replaces the referenced function call. This instruction, when executed on the DBMS machine, is trapped and recognized by a virtual data base machine monitor (VDMM) on the DBFEM. A format of the function trap instruction (FTI) is illustrated in Figure 5-28. The FTI identifies the referenced function arguments, their location on the DBMS machine and location of the SPFA input generator (SPFA/IG). The VDMM recognizes a FTI, gives control the SPFA/IG and supplies it with the location of the reference function input arguments. The SPFA/IG uses these inputs to generate a driver for the called SPFA machine.

The modification of each referenced function call can be performed by several procedures. First, the type of function must be identified.



- IDENTIFICATION
- REFERENCE FUNCTION ARGUMENTS
- # ARGUMENTS
- SPFA/IG LOCATION
- LOCATION OF SPFA MACHINE DRIVER

SAMPLE FORMAT OF A FUNCTION TRAP INSTRUCTION

FIGURE 5-28

If the function is called by application programs then these programs can be modified by loading them into the CIM array and searching for the referenced function call. Once identified, the call can be replaced by the function trap instruction. The same type of procedure can also be used if the function is called internally by the DBMS. The modified DBMS or modified software application program are included as part of the configuration resources for the DBMS/SPFA machine. Another procedure is to have the data base machine architect modify these calls directly in interactive mode on the DBFEM once a DBMS/SPFA machine is configured. In this mode, the FTI instruction is replaced directly in the main store execution of the calling instruction.

5.10.2 DBMS/SPFA Machine Selective Integration

During execution of a DBMS/SPFA machine a VDMM is used as a tool to perform the selective integration of the SPFA machine. The VDMM is included in the DBMS/SPFA Control Module.

The basic functions of the VDMM include:

(1) interface the DBMS machine architecture with a SPFA to provide a DBMS/SPFA machine,

(2) issue special resource requests for both the DBMS machine and the SPFA machine, and

(3) monitor performance of the DBMS/SPFA machine during execution.

Illustrated in Figure 5-29 is a VDMM for a DBMS/SPFA machine. In this mode the VDMM controls execution on the DBMS machine as a traditional VMM and traps all sensitive or privileged instructions.

The function trap instruction is a special case of a privileged instruction. When trapped, it notifies the VDMM that the SPFA machine

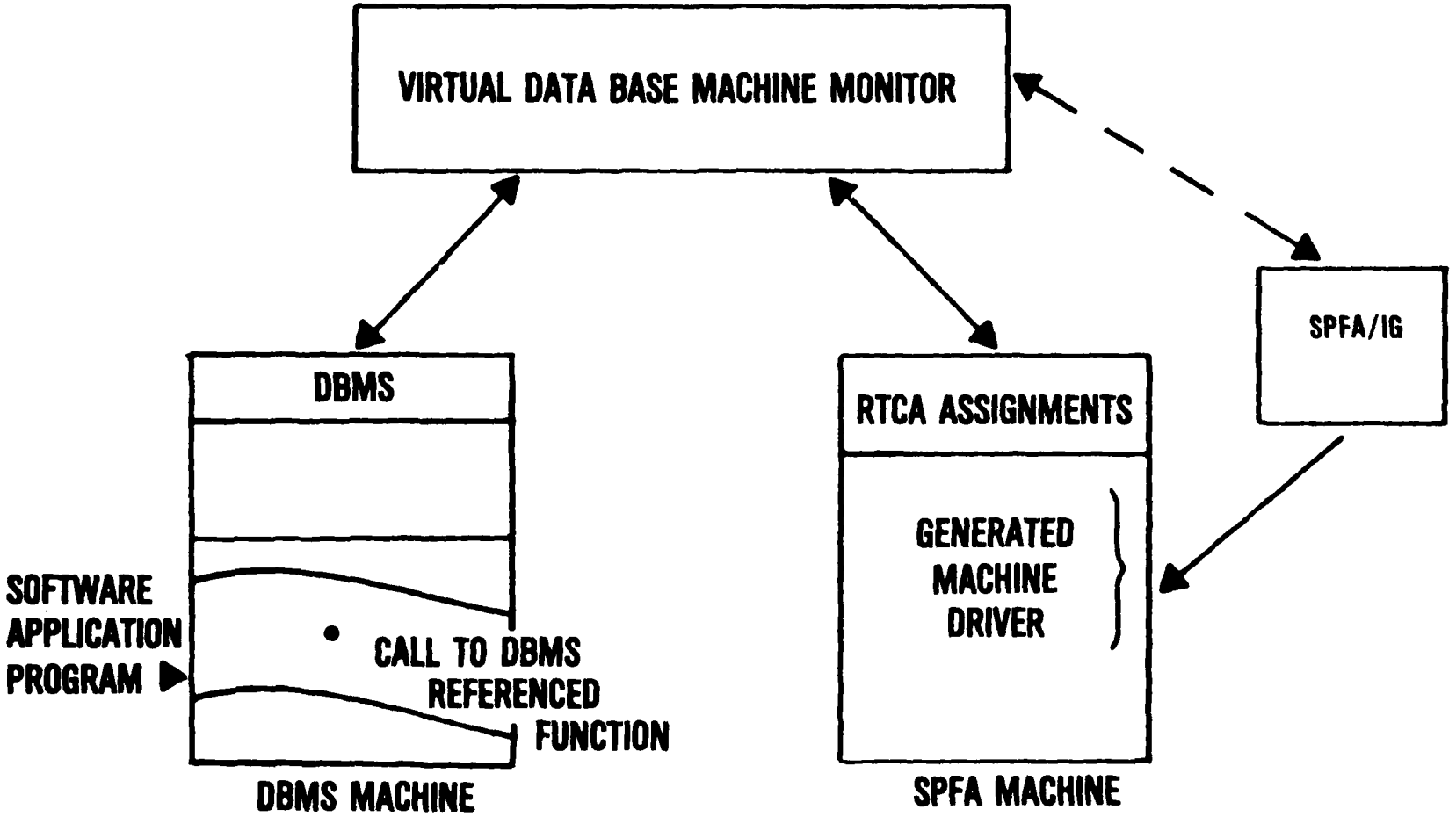


ILLUSTRATION OF A VIRTUAL DATA BASE MACHINE MONITOR

FIGURE 5-29

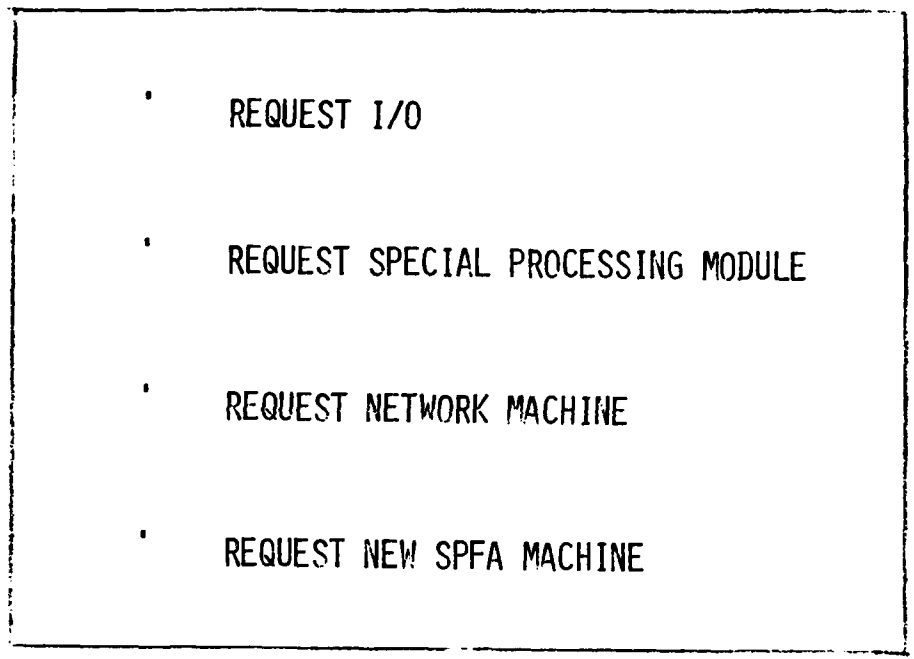
should begin execution. At this point, the VDMM gives control to the SPFA/IG and supplies it with the location of the referenced function input arguments. The SPFA/IG uses these inputs to generate a driver for the called SPFA machine. When complete, the SPFA machine is given control. The SPFA machine completes its DBMS function and control returns to the DBMS machine.

An access out of bounds or other faults are treated as execution errors. If either a DBMS machine or SPFA machine attempts such access, a fault occurs. This will cause these machines to terminate and the substitution process completes in error. A timer-run-out causes a return control to the software application program. A special resource request can also be issued by either a SPFA machine or a DBMS machine. Several types of possible special resource requests are envisioned for the DMAD Facility. An example of the type of requests are illustrated in Figure 5-30 and discussed below.

5.10.2.1 Special Resource Requests

An I/O request is issued by a machine executing on the DBFEM. This is treated as a special resource request. These requests are trapped to the VDMM. If the I/O resource is part of the configured DBMS/SPFA machine a special resource request is issued to the MCCM. The MCCM insures the resource is available and requests the resource from the Service Host Machine. The servicing of this I/O in this mode alleviates this function from the DBFEM.

A request may be issued for a special processing module that executes on the Service Host Machine. These modules are developed by the data base machine architect to perform a function in support of a SPFA machine execution. This type of request is issued by the SPFA or



SAMPLE TYPES OF SPECIAL RESOURCE REQUESTS
FIGURE 5-30

DBMS/SPFA machine during its execution. For instance, one special processing module may be the SPFA input generator (SPFA/IG). The SPFA/IG may be called, as a special processing module, to generate the machine driver for a SPFA machine. Other special processing modules may be called to analyze results generated by the SPFA machine. A special case of this type of module may be a request to a network machine.

The request to a network machine is envisioned as a request by a SPFA machine for a special tool that may be available on a unique network machine to which the DMAD Facility has access. A service like the National Software Works (NSW) (NSW79) can be utilized to provide the interface to these specialized machines.

The final type of special resource request may be from a SPFA machine for another SPFA machine. During creation of the SPFA machine the called SPFA is identified as an external call. This requires that the called SPFA must be identified in the DMCA using the Configuration Identification Machine (CIM). When a SPFA machine is called, the request passes from the DBFEM to the SHM. The SHM query formulation module formulates a query to identify the SPFA.

Once the SPFA has been identified, the originator SPFA machine may request execution of this SPFA. The request for execution then uses the resources from the SPFA identification function to configure the called SPFA machine. Inputs to the called SPFA for execution are provided by the originator SPFA. This type of request from an originator SPFA is envisioned as potentially useful as more and more SPFA's are developed and available in the DMAD Facility. During execution it is likely they will call each other to perform specialized DBMS functions.

Another request for the CIM may identify new RTCA assignments for additional executions of the same SPFA machine. For instance, when a specific execution of a SPFA machine ceases, new RTCA assignments can be identified. This request, issued directly from the DBFEM, alleviates the need to generate a new process request. This type of request can also help to automate the evaluation of several SPFA machines by permitting repetitive executions that include all possible types of RTCA timing assignments.

5.10.3 DBMS/SPFA Machine Analysis

The final substitution process function is an assessment of the SPFA machine replacing the software DBMS function. How well does the SPFA machine perform within the DBMS? Are the interfaces to it defined clearly or should they be redefined? Is the systems complexity increased or decreased as a result of moving the software function to hardware as a SPFA machine? These and other questions may be examined during this analysis. The data gathered as a result of this analysis can be used to evaluate the evolutionary impact of the SPFA machine within the DBMS machine environment using the facility. An exmaple of data that may be gathered is illustrated in the following example.

5.10.4 Substitution Process Example

The following example illustrates formation of a DBMS/SPFA machine for the substitution process. An example substitute process request is illustrated below:

PROCESS : SUBSTITUTION

MACHINE ID : S1

SPFA NAME : MERG1

DBMS MACHINE ID : P1

DBMS ID : D1

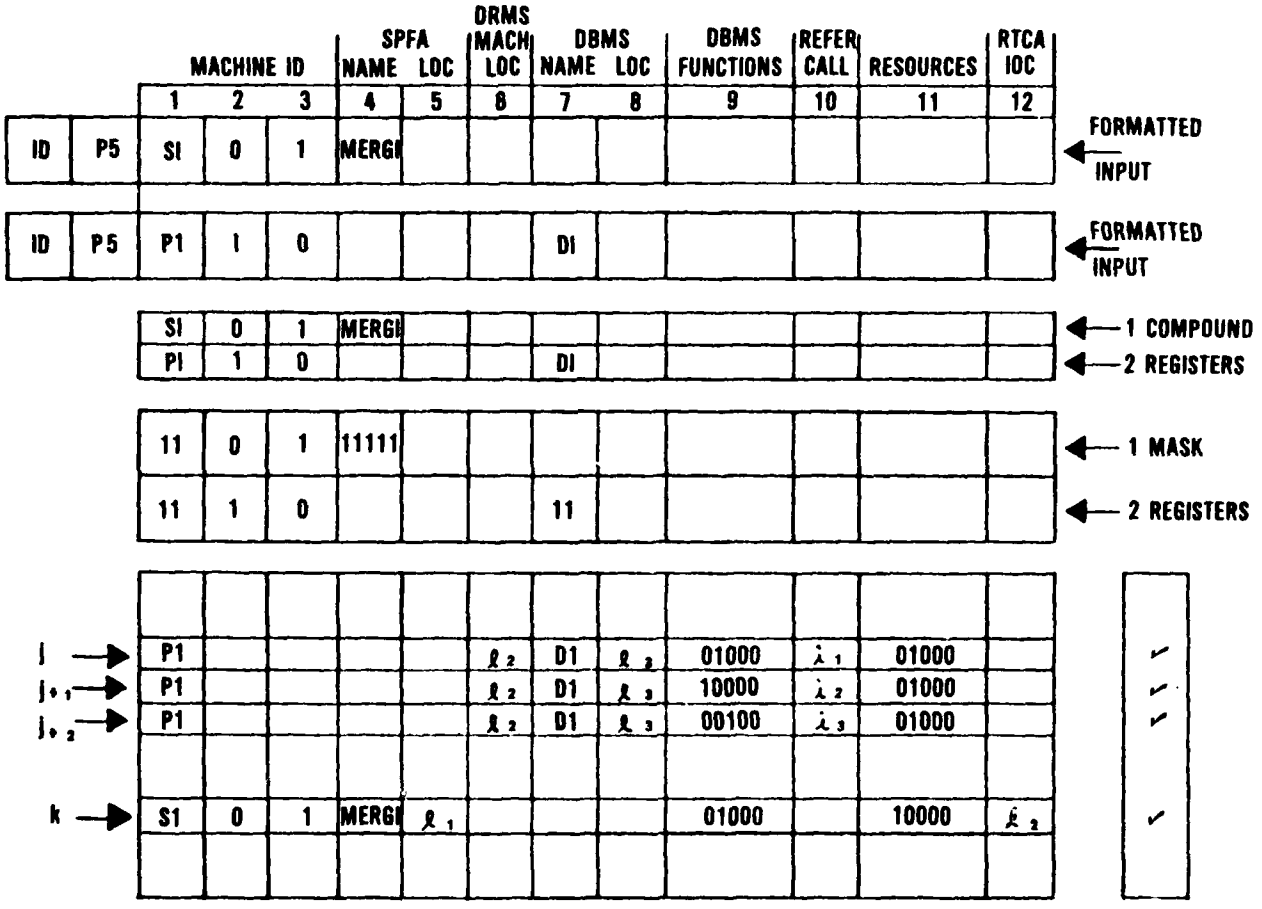
DBMS/SPFA CONTROL : EC2/VDMMP1

MODE : INTERACTIVE

SAP : @ APPROG

SPFA/IG ; @ MERG/IG

In this example, the machines are the S1 SPFA machine and the P1 DBMS machine. A DBMS "D1" is supported on the P1 machine. The DBMS/SPFA Control Module "EC2/VDMMP1" is requested. Finally, a software application program APPROG is identified. Two sets of input to the DMCA, illustrated in Figure 5-31 are generated. Next, separate substitution queries are generated. The first substitute process query loads the Comparand Register with the input for the SPFA machine identification. Field 1,3, and 4 identified by Mask Register 1 are used as the search criteria. An equal to comparison produces entry k and the SPFA machine (Figure 5-31). A similar procedure is performed for the DBMS machine. Assume entries j, j + 1, and j + 2 represent the complete DBMS machine. The D1 DBMS supports DBMS functions f_1, f_2, and f_3. Entry k is combined with entries j + 1 and j + 2 to produce the DBMS/SPFA machine. The resources needed to configure this machine are combined by OR'ing the resource field, filed 11. This produces r_1 and r_2. The referenced DBMS function is the f_2 DBMS function. Its required set of interfaces have been defined and are defined at location 1. This function uses this location to obtain these interfaces.



DBMS/SPFA MACHINE IDENTIFICATION USING A DATA BASE MACHINE ARCHITECTURE CONFIGURATION ARRAY (DMCA)

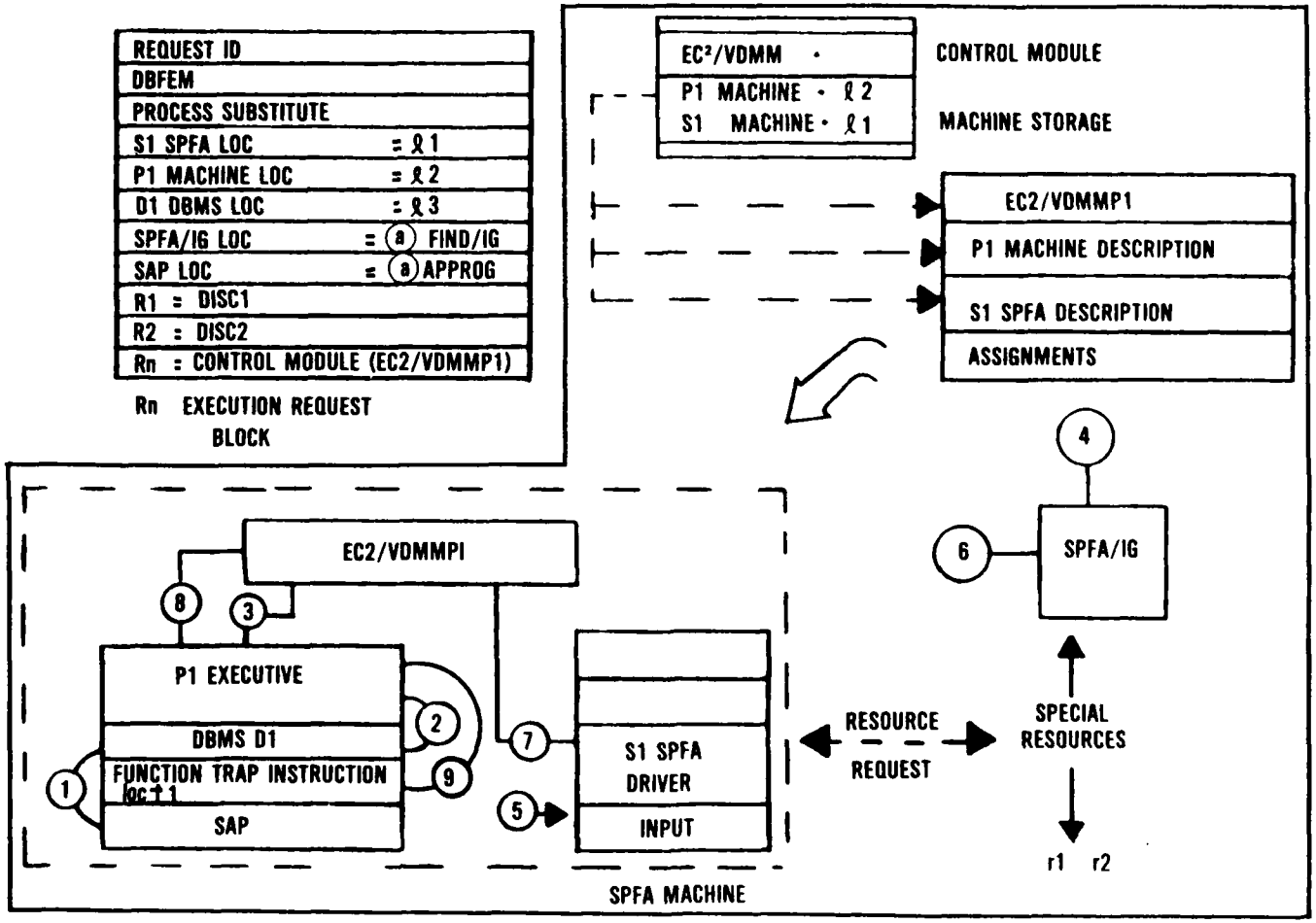
FIGURE 5-31

Next, the DBMS machine pre-processing is performed. First a function trap instruction is generated. The instruction identifies the location of the SPFA/IG as @ FIND/IG which was defined in the input process request. The FTI replaces calls to the function directly. This modification can be done directly on the DBFEM, once the DBMS/SPFA machine is configured.

The configuration of the DBMS/SPFA machine needs the resources identified in the DMCA. The location of the DBMS machine "YYY", the location of the D_1 DBMS "AAA", and the location of the SPFA machine "XXX", illustrated in Figure 5-31 are part of these resources. These resources are used to develop the Execute Request Block (ERB), as illustrated in Figure 5-32. The ERB identifies the Execution Control Module on the DBFEM as the EC2/VDMMP1 Control Module. The EC2/VDMMP1 Control Module initializes the P1 and S1 machine. Resource r_2 is assigned to the P1 machine while r_1 is assigned to the S1 SPFA machine. The P1 machine's main store is initialized with the D1 DBMS. The software application program APPROG is also initialized on the P1 machine. The SPFA/IG is configured as a special resource. The l_1 location on the DBMS machine is replaced directly with the function trap instruction.

Execution begins at the first executable location of the APPROG software application program. This program executes normally as it would on the P1 machine. This program calls on the services of the D1 DBMS (1). The D1 DBMS provides all DBMS support with the exception of the function f_2. This function is provided by the S1 SPFA.

When the function trap instruction is executed, on the DBMS Machine, control passes to the EC2/VDMMP1 (2,3). The FTI identifies the location



EXAMPLE OF DMAD FACILITY TRANSFORMED INTO A DBMS/SPFA MACHINE

FIGURE 5-32

of the SPFA/IG and location l_1 as the required argument list. The EC2/VDMMP1 passes control to the SPFA/IG (4). Once the SPFA/IG has control it uses the arguments from location l_1 to generate a machine driver for the SPFA machine. This driver is targeted into the main store location, by the VDMM, where the SPFA machine begins execution (5). When the SPFA/IG completes this driver, control returns to the EC2/VDMMP1 (6). The EC2/VDMMP1 now passes control to the SPFA Machine (7). The SPFA machine begins execution using the input from the machine driver. This machine continues execution until it completes its DBMS function. When complete, control returns to the EC2/VDMMP1 (8). The EC2/VDMMP1 returns control to the instruction in the software application program following the referenced function call (9,10). The software application program continues execution.

5.11 SUMMARY

This chapter establishes a set of procedures to illustrate how a DBMS function is chosen and how a SPFA is developed using the DMAD Facility. Various types of procedures can be utilized during this development. The types of tools and components available in the DMAD Facility permit a data base machine architect to tailor the development of one or more SPFA's to meet specific application requirements. During this development, data is accumulated to help decide which SPFA is a candidate to be a hardware prototype. The total utilization of the facility in this manner provides a localized environment where candidate DBMS functions are chosen and SPFA's can be developed, executed and evaluated.

CHAPTER VI

TOOLS/COMPONENTS USED TO DEMONSTRATE A SPECIFIC IMPLEMENTATION OF DMAD FACILITY

6.1 INTRODUCTION

Described in this chapter is a set of tools/components used to demonstrate a specific implementation of the DMAD Facility. These tools/components consist of:

(1) Nanodata QM-1 Microprogrammable Computer,

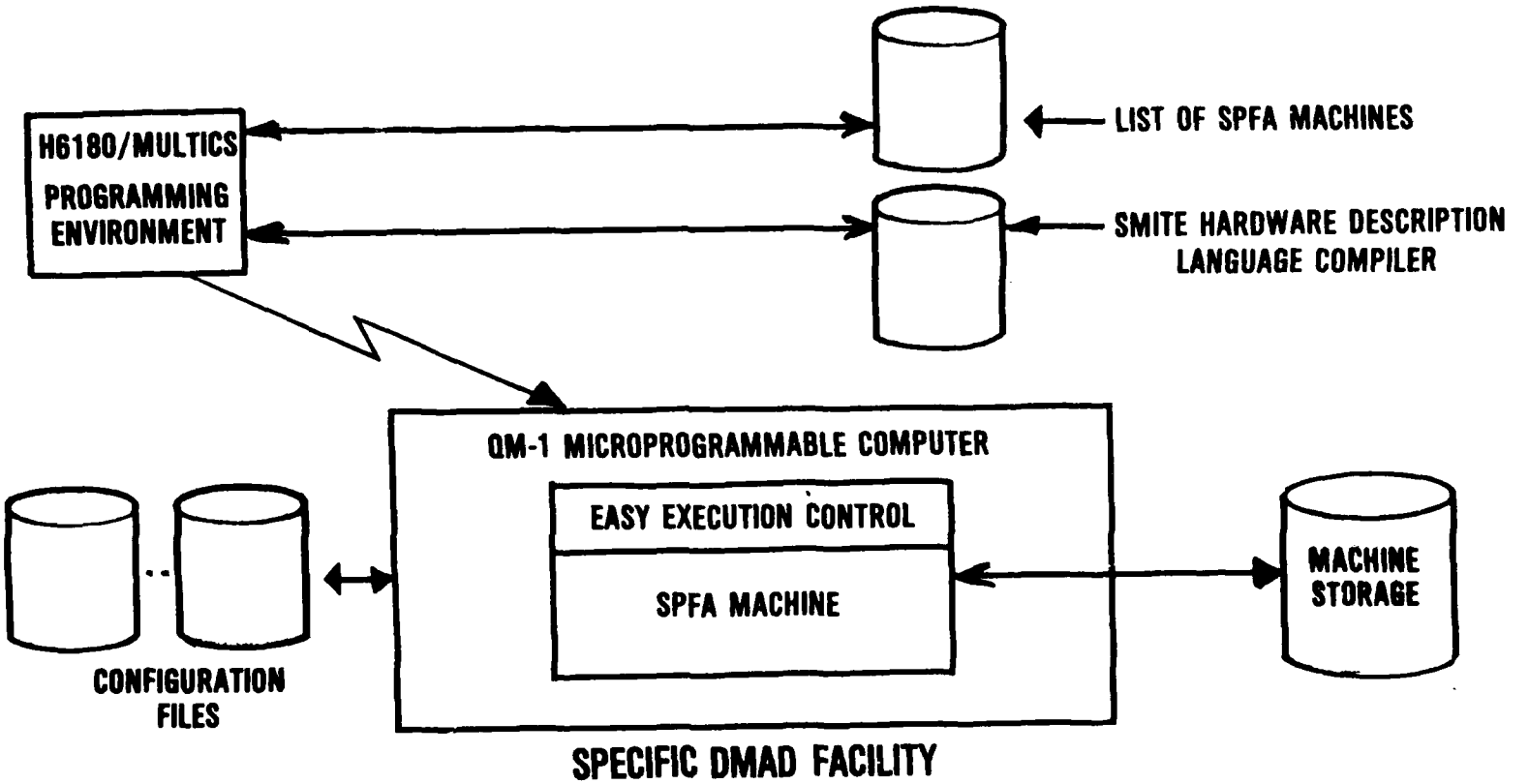
(2) SMITE hardware description langugage, and

(3) the MULTICS/H6180 Programming Environment.

6.2 ORGANIZATION OF THE SPECIFIC DMAD FACILITY

The organization of the specific DMAD Facility is illustrated in Figure 6-1. This facility will be used to actually demonstrate SPFA development. The H6180 MULTICS Programming Environment maintains the SMITE hardware description language compilier along with a list of available SPFA machines. The SMITE hardware description language is used to create and/or modify descriptions of these SPFA machines. Once a machine is created it is down-loaded from the MULTICS environment directly to the Nanodata QM-1. The EASY execution control module is used to configure each SPFA machine using the QM-1. A set of configuration files are established that provide the components necessary for configuration. Once configured the SPFA machine is available for testing and evaluation.

The EASY Execution Control, on the QM-1, provides an extensive



COMPONENTS UTILIZED FOR SPECIFIC DMAD FACILITY IMPLEMENTATION

FIGURE 6-1

set of interactive dynamic testing and debugging commands (EASY76) to control the execution of each SPFA machine. Once a SPFA machine has been fully tested several performance evaluation options are available. A realization timing control array (RTCA) is prepared for each SPFA machine. Timing values assigned from the RTCA are configured as part of each SPFA machine and are used to generate performance evaluations for each machine. The selection of each of these tools/components and their specific utilization in this facility is now described.

6.3 QM-1 MICROPROGRAMMABLE COMPUTER AS EXECUTION MACHINE

In the past, the QM-1 microprogrammable computer has been used, primarily, as a tool to develop emulations of various computer systems. However, in this implementation it is shown that it can be utilized as a tool for the development of new data base management hardware architectures. In this role, the QM-1 is used to execute the DBMS SPFA machines. The QM-1 is particularly attractive for this development because of the following capabilities:

(1) the actual architecture of the QM-1 permits it to emulate a variety of types of special purpose function architectures,

(2) the QM-1 can execute descriptions of special purpose function architectures described in the SMITE hardware description language, and

(3) the QM-1 provides a set of highly specialized testing and debugging tools for the data base machine architect using the EASY execution control module.

The actual architecture/interfaces of the QM-1 are illustrated in Figure 6-2 and are described below.

6.3.1 Use of QM-1 Main and Control Stores to Execute SPFA Machines

The QM-1 possesses two levels of microprogrammed control using three memories:

(1) Main Store,

(2) Control Store, and

(3) Nano Store.

Instructions are entered and executed at each level of store. Main store contains instructions of the target machine (i.e. the machine being emulated). These instructions are interpreted and executed by micro-program in control store under vertical control. Finally, MULTI micro-instructions in control store are,in turn, executed by (and defined by) nanoprograms in nanostore under horizontal control.

Illustrated in Figure 6-3 is how the two level control store is used to execute a SPFA machine. Each compiliation of a SPFA description produces a set of MULTI micro-instructions. These instructions are used to configure each SPFA machine and are loaded in the first level control store of the QM-1. The set of registers described for each SPFA machine are also included in control store. Next, a driver for each SPFA is established by the data base machine architect. This driver identifies to the SPFA machine a set of input and output interfaces which it needs to perform its DBMS function.

The second level control store, called nano store, uses nano code to interpret the set of MULTI instructions in the first level control

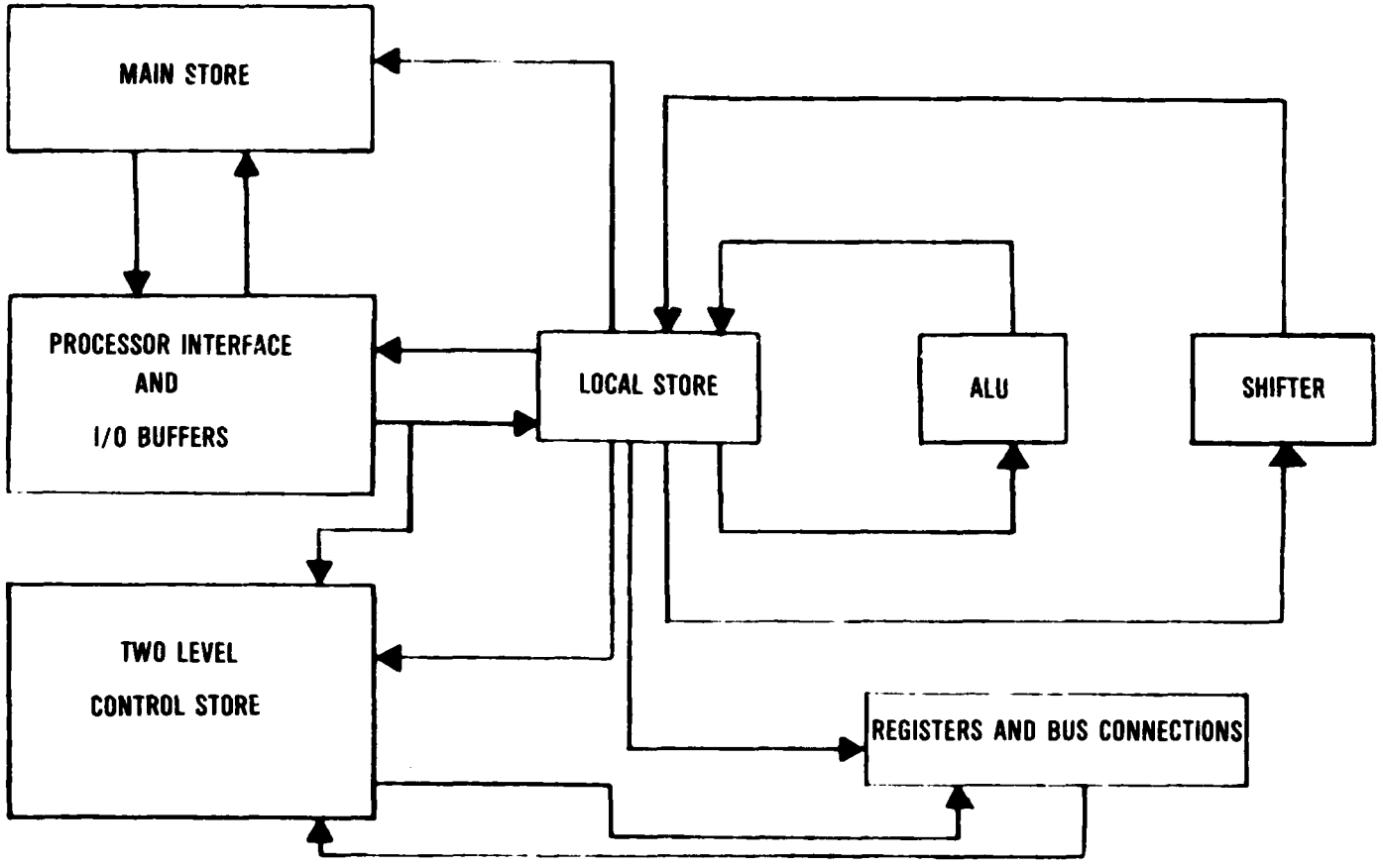
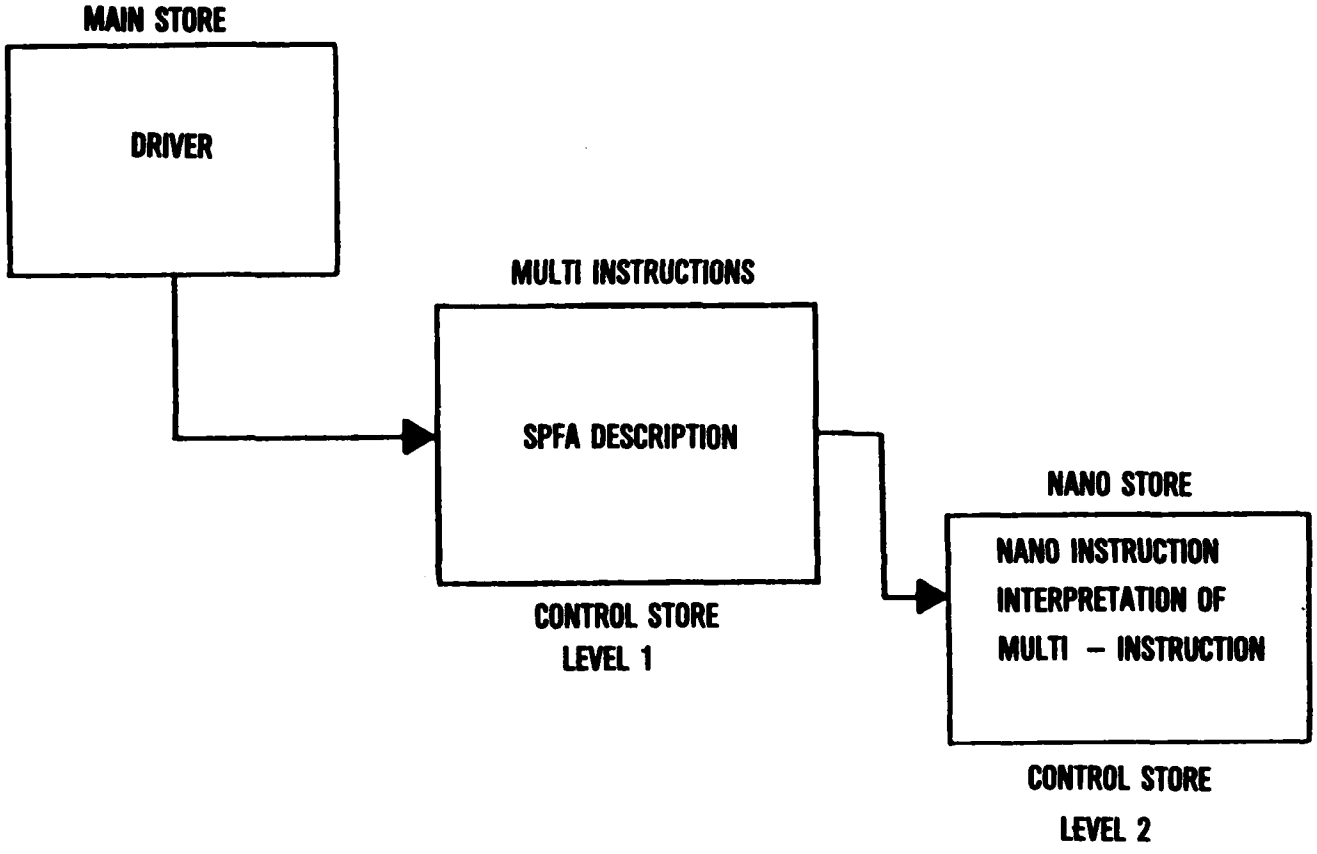


ILLUSTRATION OF ARCHITECTURE OF QM-1 MICROPROGRAMMABLE COMPUTER

FIGURE 6-2



UTILIZATION OF QM-1 TWO LEVEL CONTROL STORE HIERARCHY FOR SPFA MACHINES

FIGURE 6-3

store. The nano store can be used to provide additional flexibility in the description of a SPFA if unique features cannot be described in SMITE. For instance, at this level a new MULTI instruction can be created by the data base machine architect to perform a unique architectural construct of a SPFA. This instruction is then interpreted by a set of nano code to perform its intended function. This flexibility is extremely important to the DBMA to insure that a SPFA machine is being correctly emulated and provides the capability to take advantage of both vertical and horizontal micro-code features on the QM-1.

The other features of the QM-1 used to control the execution of SPFA machines, illustrated in Figure 6-2, are:

(1) Local Store,

(2) Arithmetic-Logic Unit, and

(3) the Shifter.

Local store is a bank of 32 18 bit registers. Some of these registers have special uses; for instance, the microprogram counter and the micro-instruction register are kept in local store.

The Arithmetic-logic Unit can be controlled to perform all of the 16 logical (Boolean) operations, as well as common arithmetic operations, upon two 18 bit operands. A 16 bit mode permits the two input operands to be sign-extended from 16 to 18 bits so that the operation of the ALU need not be changed when dealing with 16 bit data values. Also, a decimal control facilities decimal arithmetic.

The 18 bit combination logic shifter is capable of performing both left and right logical and circular shifts, as well as right

arithmetic shifts. Also, double precision (36 bit) left and right logical, arithmetic and circular shifts can be specified.

6.4 SMITE HARDWARE DESCRIPTION LANGUAGE TO DESCRIBE SPFA'S

The actual description of a SPFA is developed using the SMITE hardware description language. This is a high level language that is intended to describe computer architectures using a well disciplined, top-down structured approach. A procedure to describe a SPFA in SMITE within the H6180 MULTICS Programming environment is illustrated in Appendix A. SMITE is utilized in this facility because specific features make it particularly attractive for describing SPFA's.

6.4.1 SMITE Features for SPFA Descriptions

These features include:

(1) ability to specify parallel architecture constructs,

(2) ability to control execution paths of the SPFA, and

(3) ability for timing specification.

A parallel architecture construct for a SPFA is specified, in SMITE, using a parallel context block. This block permits a data base machine architect to define a series of operations that execute simultaneously and asynchronously with all other statements in the parallel context. These processes start execution at the same emulation time and the parallel context completes execution when the last operation finishes. The actual implementation of parallelism is executed using the EASY emulator control module. Operations within a parallel context block are executed as concurrent subtasks. The timing clock for a parallel context block is updated when the last operation completes to the maximum value of the sub-operations.

The ability to control execution paths for a SPFA is provided, in SMITE, by several types of statements;

(1) the IF Statement,

(2) the Case Statement, and

(3) the DO While Statement.

These statements identify control for the SPFA machine and enable selection of appropriate execution paths during its execution. These statements also provide the capability to implement a top-down structured approach for each SPFA machine description. With this approach, each description also serves as a document of the architecture for each machine.

The timing specification is important because it allows the performance of each SPFA to be obtained. This performance can be used as a basis of comparing two or more SPFA's with different architectural constructs that perform the same DBMS function.

6.4.2 Timing Procedure for SPFA Machines

The following procedure has been developed to conduct performance timings for each SPFA machine. This procedure uses a realization timing control array (RTCA) to define a set of operations from each SPFA description. These operations consist of one or more SMITE statements that describe an operation group performed by the SPFA machine. Once these operations groups are defined, one or more realization assumptions are selected. Each realization assumption is used to assign a time assigned to a specific operation. A set of various assignments of times for 1 through n operation groups is illustrated for a RTCA in Figure 6-4.

CHARACTER-ISTICS / SPFA OPERATIONS	REALIZATION ASSUMPTION (RA1)	REALIZATION ASSUMPTION (RA2)	. . . . .	REALIZATION ASSUMPTION (RA n)
OPERATION 1	t(1,1)	t(1,2)		
• •				
OPERATION n	t(n,1)			t(n,m)

OUTLINE OF SPFA REALIZATION TIMING CONTROL ARRAY

FIGURE 6-4

These times are designated as t(n,m) where n is the operation group number and m is the realization assumption. The times for an operation group may vary according to the realization assumption.

Once an RTCA is formed it is used to assign times to each operation group that is described as part of a SPFA machine. These timings are incorporated using the SMITE IN statement.

An example of how this statement is utilized is shown below:

OPERATION TYPE	Statement #	
Non-executable	1	DEFINE XX 15:0 REGISTER,
Non-executable	2	DEFINE YY 15:0 REGISTER,
Information Transfer	n	IN "expression" XX ← YY;

In the above description two registers are specified, XX and YY. Each register is 16 bits. The basic operation, identified at statement n, is to transfer the contents of YY to XX. The IN "expression" statement designates in "expression" the time necessary to perform the operation designated by statement n. This time is assigned to the transfer operation in the RTCA.

6.4.2.1 RTCA Assignment

The time assigned to complete the operation is based on its implementation in hardware. For instance, assume a realization assumption identifies transfer circuit type and hardware technology type. The time assigned to an information transfer operation can then be based on one of the technologies and transfer circuitry shown in the following table:

	Jam Circuit	Clear and Copy Circuit
Technology A	J1	J2
Technology B	J3	J4

For instance, if a realization assumption consists of technology A and the choice of a jam circuit for an information transfer operation, then the time J1 is entered in the Realization Timing Control Array for this operation. The same operation may be assigned different times depending on changes to the realization assumption. These changes can be made to analyze a SPFA's overall sensitivity to a hardware characteristic change. Thus, for the same operation, times J1, or J2, or J3, etc. may be assigned.

This procedure provides the capability of assigning variations in times for each operation in the SPFA. These variations can be made to assess relationship between performance and unique application criteria. For instance, the performance of an operation may be sacrificed for higher reliability factors of the hardware characteristics included in the realization assumption. Thus, performance verses reliability tradeoffs can be studied. In the same manner, other types of tradeoffs can be studied.

An example of how this procedure can be used for a SPFA machine is now described. A set of SMITE statements that identify a processor for the COMPARISON operation is illustrated in Figure 6-5. The operation is based on a comparison element needed as part of Batcher's odd-even merge network (BATC68). This processor is needed during execution of a [illegible] that uses the odd-even merge network. The processor

```
COMPARISON PROCESSOR  ( I 1, I 2 );

   DECLARE  I 1   <15:0>  REGISTER,
            I 2   <15:0>  REGISTER,
            TEMP  <15:0>  REGISTER;

   IN TIMER-VALUE    4   BEGIN;
            IF   I 2 < I 1
            THEN BEGIN ;
            TEMP ← I 2  ;
            I 2 ← I 1  ;
            I 1 ← TEMP ;
            END ;
            END IF ;

   COMPARISON END :
```

FIGURE 6-5 A COMPARISON OPERATION FOR A SPFA MACHINE

receives two inputs and provides, as output, the minimum on one line and the maximum on another line. The time assigned to the operation group is defined as IN TIMER-VALUE (4). TIMER-VALUE is an array that has been declared as part of the SPFA machine description. This array is identified by declaring the following type of statement in the SPFA description.

TIMER-VALUES (subscrip 1 : j) bit size REGISTER ;

This statement provides space to enter a times from the RTCA during configuration of the SPFA machine. A set of registers are declared in the SPFA description, from 1 to J, that correspond to each of the operation groups to be timed. An example of this declaration is:

TIMER-VALUES (1:10) <15:0> REGISTER ;

This declaration identifies an array of 10 registers, each 16 bits in length. In order to assign the time for the COMPARE operation, a realization assumption is chosen prior to execution of the SPFA machine. The set of characteristics in the realization assumption are used to compute the timings for this operation. The time assigned to the COMPARE operation, at TIMER-VALUE (4) is based either on a specialized hardware component that performs this operation or is assigned the time computed for a logic circuit to complete the operation.

The timing for each SPFA machine is determined by accumulating the time whenever each of the designated operation groups are executed. This time accumulated directly on the QM-1 in a timing control clock designated as a local storage register.

During the execution of a machine the timing for any operation can be varied. These changes are made to use the facility to illustrate the

sensitivity of a specific operation or operation group of the machine. If this machine should be chosen for implementation then the facility can be used to designate highly sensitive operations which may need unique implementation consideration.

6.4.2.2 RTCA Calculations

This timing analysis provides a flexible approach to help determine the performance of a SPFA. The calculations used to assign timings to each operation group identified are represented as:

Let OGn (s:s+r) = operation group consisting of statements s to s + r,

and $t_{(o,s,m)}$ = time for operation o, at statement s, using realization assumption m,

Then the time for an operation group (OGn) is:

$$t_{(OGn,\ s:s+r,m)} = t_{(o,s,m)} + t_{(o,s+1,m)} + \ldots + t_{(o,s+r,m)} \qquad (6.1)$$

If the time for this operation group is computed using the time from an integrated circuit to perform the operation, this time can be represented as:

Let OG (s:s+r) = operation group consisting of statements s to s+r

m = realization assumption m, consists of integrated circuit (IC) that can perform operation group (OG)

and t_{IC} = time for integrated circuit to perform operation operation group (OG)

then the time assigned to operation (OG) is:

$$t_{(OG,s:\ s+r,m)} = t_{IC} \qquad (6.2)$$

6.4.3 Identification of Operations Used for SPFA Machines

The following types of operations have been selected as the types of operations needed by SPFA machines. These operations are described in SMITE using one or more statements. These operations consist of:

(a) transfer of information,

(b) indicator set,

(c) control of flow (selection), and

(d) compare.

The transfer of information operation is used to transfer information within a SPFA machine. This information is transferred among registers that are declared as part of each machine. For instance, the following declaration:

```
DECLARE   X-INPUT-REG     <15:0>   REGISTER;
          Y-INPUT-REG     <15:0>   REGISTER;
```

defines two 16 bit registers called X-INPUT-REG. and Y-INPUT-REG.

Illustrated in Figure 6-6 is the transfer of information from the i th bit of X-INPUT-REG to the i th bit of Y-INPUT-REG. This transfer in SMITE is described as follows:

```
Y-INPUT-REG <X:i> ← X-INPUT-REG <X:i> ;
```

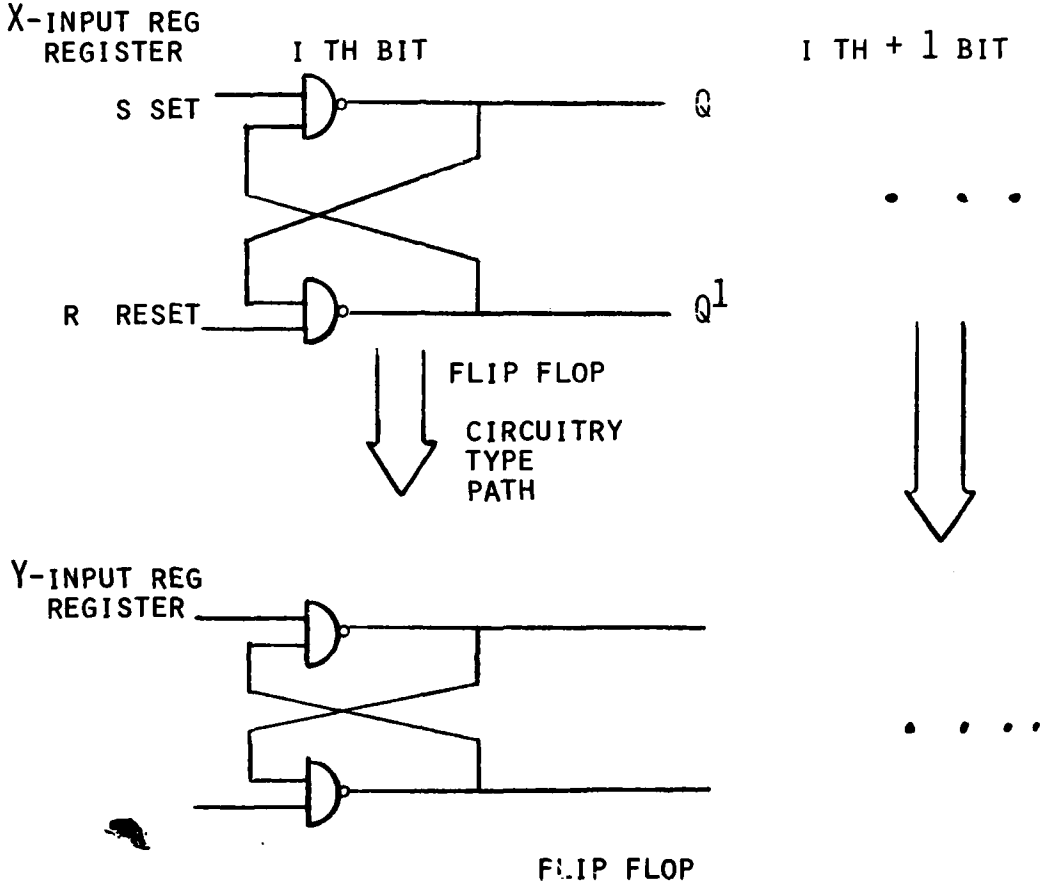
For a transfer of all bits from X-INPUT-REG to Y-INPUT-REG, the following notation is used:

```
X-INPUT-REG ← Y-INPUT-REG ;
```

Several factors are examined to help determine a timing for this operation:

(a) hardware technology used in the transfer, and

(b) type of operation.



EXAMPLE OF AN INFORMATION TRANSFER OPERATION

FIGURE 6-6

The type of information transfer operation is based on the SMITE description. Two types of information transfers that can be specified in SMITE and included in a SPFA machine description are:

(1) a simple transfer, or

(2) a compound transfer.

A simple transfer, in SMITE, is of the form:

A ⟵ B ;

This statement implies the contents of B is transferred to A. In actual hardware this implies the SPFA has available two registers and a path for transferring information from one register to another.

A compound transfer is of the form:

A ⟵ A + B ;

This type of transfer performs one or more arithmetic or logic operations followed by an information transfer operation.

The indicator set operation is used to set or reset a condition in a SPFA machine. Each machine is described with a set of these indicators. This condition is set by transferring a zero or one into the indicator. These conditions represent a TRUE or FALSE state in the machine. An example of this operation is:

X-INPUT-REG ⟵ 1 ;

The control of flow operations are used to determine the execution path of the machine. This operation evaluates a logical expression to a condition that either sets or resets an indicator. This indicator is tested and one or more operation groups are selected to execute.

There are several types of statements used to control flow within SPFA machines:

Example One: "DO WHILE" expression

OPERATION GROUP 1

The DO WHILE statement permits the operation group following the DO WHILE declaration to continue execution as long as the expression evaluates to a TRUE condition.

Example Two: IF expression

THEN

OPERATION GROUP 1

ELSE

OPERATION GROUP 2

The IF statement determines which operation group is executed following the IF declaration by evaluating the expression to a TRUE or FALSE condition. If the expression evaluates to a TRUE condition, then operation group 1 is executed. If it evaluates to a FALSE condition, then operation group 2 is executed.

The compare operation is used to determine if two inputs are equal, greater than or less than. The operation then sets or resets an indicator based on the compare condition. The setting and testing of this condition determines the execution path of the machine.

Several other operations may be described as part of a SPFA machine. These operations are based on the specific DBMS function the machine performs. For instance, the shift operation may be needed by a SPFA machine. The assignment of time to these operations permits an overall assessment of performance of the SPFA machine. The performance of two or more SPFA

machines with different architectures can be compared using this procedure. Tradeoffs between these machines, when each performs the same DBMS function, may be examined. Sensitivity to changes in their architectural features can be examined using this procedure. Also, projected performance improvements for SPFA machines due to new technology advances can be measured.

6.5 SUMMARY

This chapter has outlined a series of components used to demonstrate a specific implementation of the Data Base Machine Architecture Development Facility. These components were chosen because they represent the type of capability needed in a DMAD Facility. The QM-1 microprogrammable computer is chosen because its specialized capabilities enable it to execute as a SPFA machine's. SMITE is chosen because it is a highly specialized hardware description language. It is shown how SMITE capabilities can be used to describe unique SPFA's. The H6180 MULTICS programming environment provides specialized capabilities to describe SPFA's. A procedure is described to develop and evaluate SPFA machines using this specific facility. The next chapter describes a procedure to illustrate the feasibility of using the tools/components of the specific facility to develop SPFA's.

CHAPTER VII

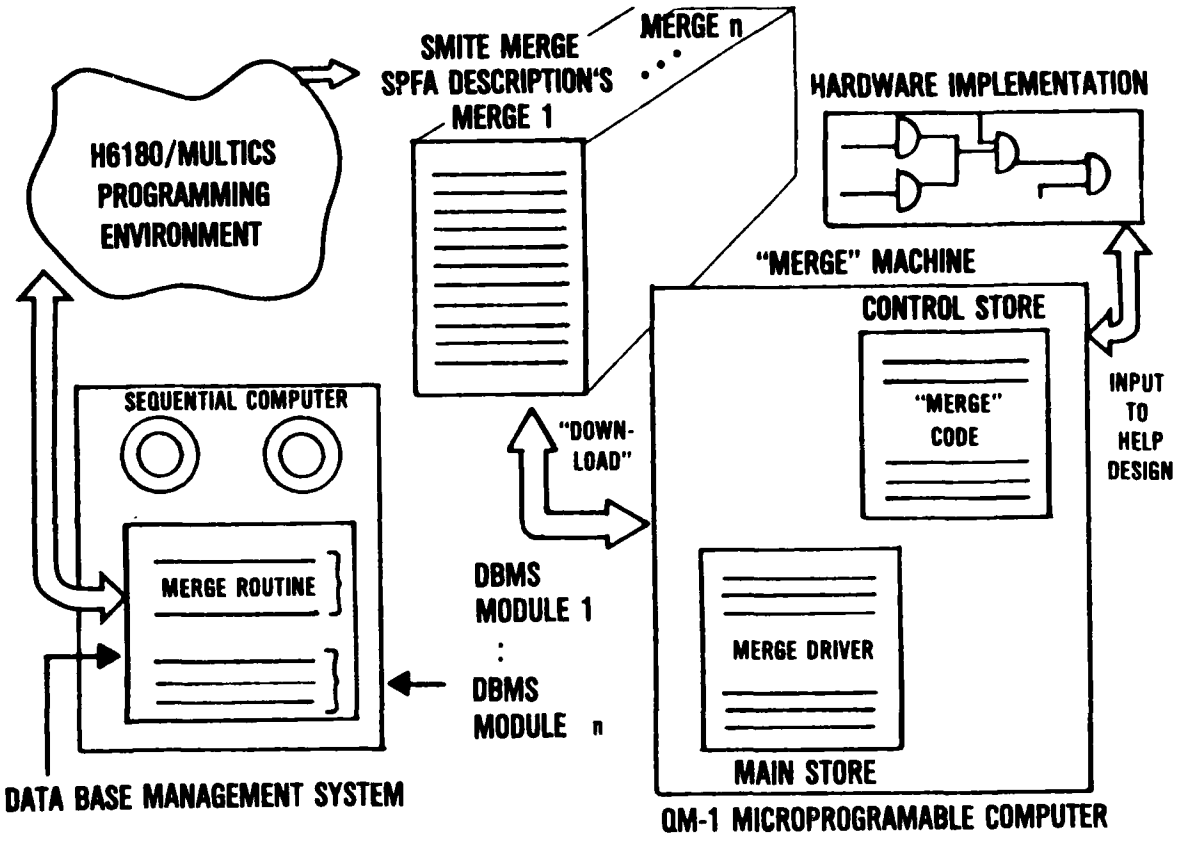
DEMONSTRATION OF FEASIBILITY OF UTILIZING A SPECIFIC DMAD FACILITY TO DEVELOP SPECIAL PURPOSE FUNCTION ARCHITECTURES

7.1 INTRODUCTION

Described in this chapter is an illustration of how the specific facility described in CHAPTER VI can be used to develop special purpose function architectures. An illustration of this demonstration is shown in Figure 7-1. First, a merge DBMS software routine is assumed to be selected as a candidate DBMS function to be moved into hardware. Next, several SPFA's were selected to perform this function and were described using the SMITE hardware description language. These descriptions were compiled in the H6180/MULTICS programming environment to produce QM-1 Multi machine loadable code. This code was transferred from the H6180/MULTICS programming environment to the QM-1. These SPFA's were configured on the QM-1 as merge SPFA machines. The SPFA machines were then tested and evaluated on the QM-1 using the facility to generate various types of timing data. Next, it is shown how the facility is used to study the machines, modify the machines and tailor their development to specific applications. Discussed in the remainder of this chapter are specific details of this demonstration.

7.2 USE OF FACILITY TO ESTABLISH MERGE AS CANDIDATE DBMS FUNCTION

In order to demonstrate SPFA development it is necessary to illustrate how the facility can be used to help choose the merge function as a candidate DBMS function to be moved from software to hardware. This process is illustrated by the following example. First, assume that a



MERGE SPFA DEVELOPMENT UTILIZING A QM-1 MICROPROGRAMMABLE COMPUTER

FIGURE 7-1

DBMS machine is emulated to execute on the QM-1 microprogrammable computer. Several emulations of DBMS machines have been developed and executed on the QM-1 microprogrammable computer (AGRA76). For instance, the NOVA, IBM 360, and PDP 11/45 have been emulated. Each of these machines can support a data base management capability. Some examples are IMS, TOTAL, SYSTEM 2000 and others.

Next, assume the DBMS machine executes in the facility as illustrated in Figure 7-2 and, for this discussion, the data base management system it supports is called D1. Configuration files in the facility are used to configure the DBMS machine with a set of system support software, the D1 data base management system and application programs. The application programs utilize the D1 DBMS and its DBMS functions. In order to help choose which DBMS functions are candidates to be moved into hardware, the facility is utilized to examine the type of DBMS functions required, frequency of being called and their complexity within the data base management system. This examination is conducted by gathering data for the following:

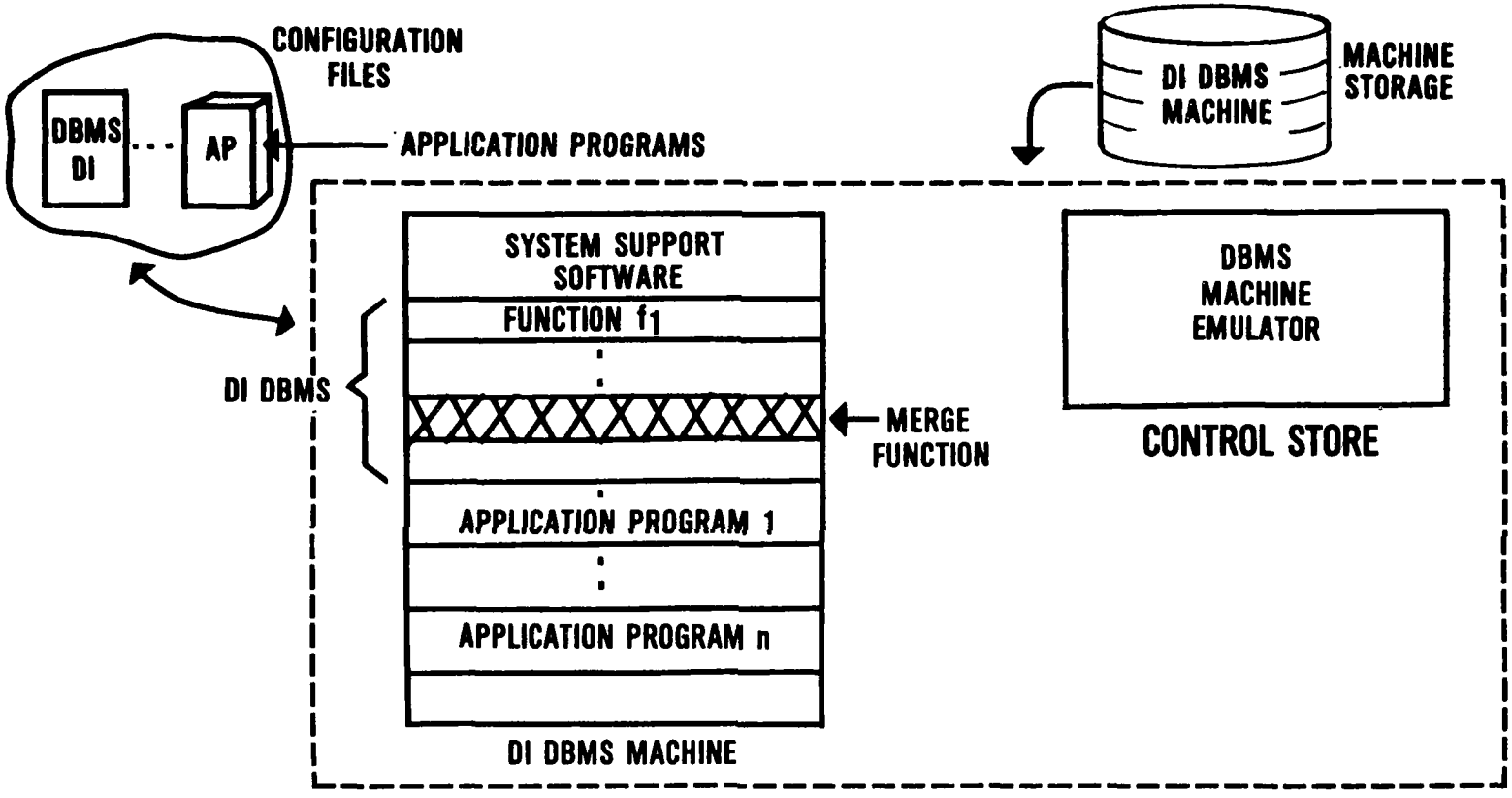
(1) statistics on utilization of a specific DBMS function for an application,

(2) identification of interfaces to a DBMS software function,

(3) dynamic software testing of a specific DBMS software module, and

(4) quality of the DBMS functions.

A procedure to use the facility to collect this type of data is illustrated in Figure 7-3. First, a set of DBMS function modules are identified. Next, all entrances to each DBMS software function module



CONFIGURATION OF FACILITY USED TO COLLECT UTILIZATION DATA

FIGURE 7-2

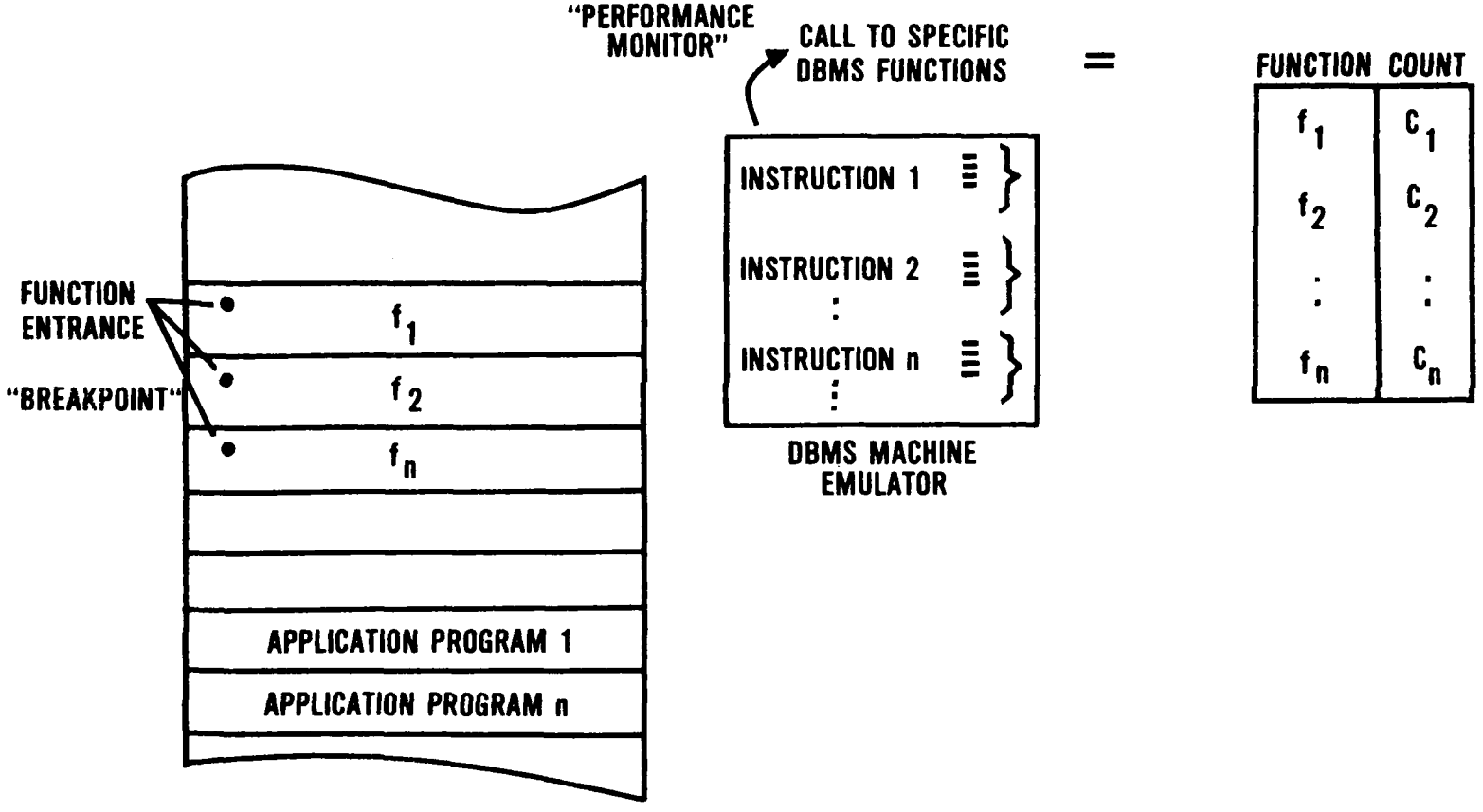
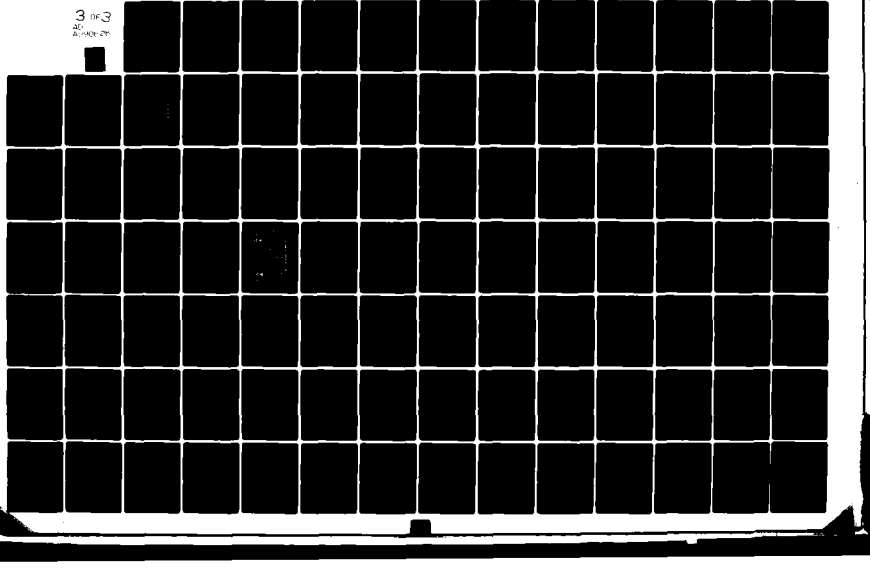


ILLUSTRATION OF PROCEDURES TO COLLECT DBMS UTILIZATION DATA USING FACILITY

FIGURE 7-3

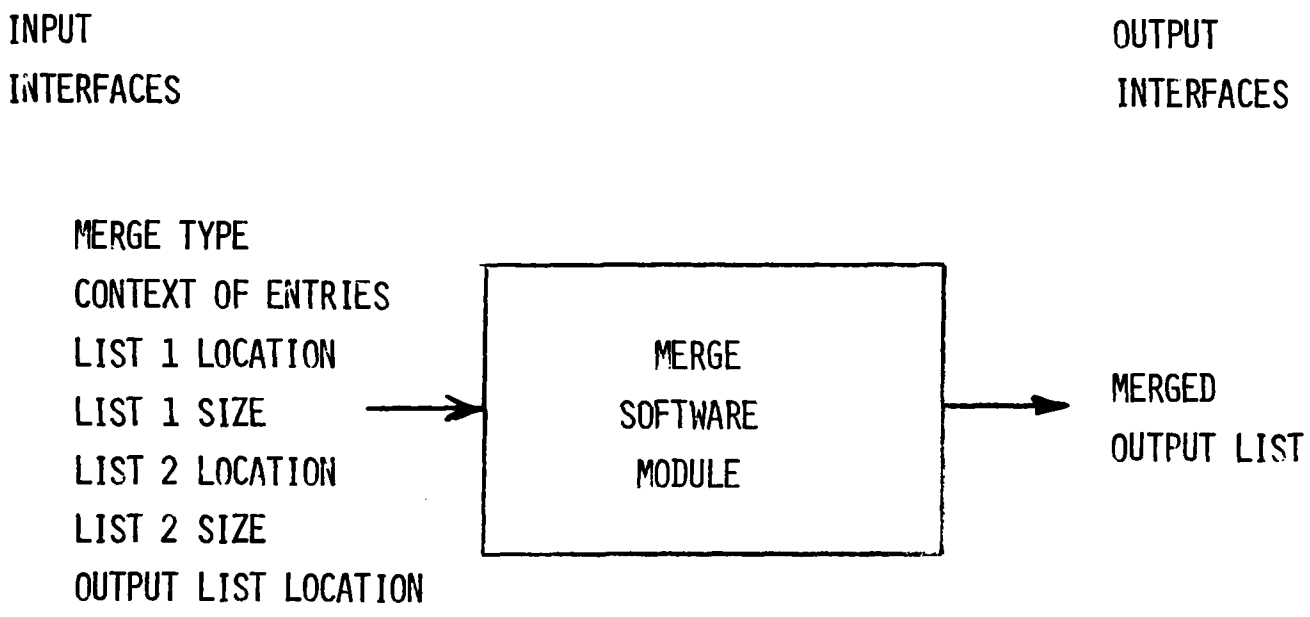
AD-A090 826 ROME AIR DEVELOPMENT CENTER GRIFFISS AFB NY F/G 9/2
THE SPECIFICATION OF A DATA BASE MACHINE ARCHITECTURE DEVELOPME--ETC(U)
JUL 80 R A LIUZZI
UNCLASSIFIED RADC-TR-80-256 NL



are identified. The facility is then used to establish breakpoints at some or all of these locations. A set of applications programs are selected and the DBMS machine is allowed to begin execution. Whenever the breakpoints are encountered the DBMS machine interrupts execution to permit a user to examine both the state of the function and machine. During this examination utilization counts, function type and other pertinent data are established. This process is repeated for one or more sets of application programs.

Assume, for this example, a high frequency of calls are made from the application programs for the merge DBMS function. This software module selectively merges input data lists into a single output list. Once the function is identified the facility can be used to isolate the actual merge software module from the remaining parts of the D1 DBMS. This isolation permits using the facility to establish the complexity of the module within the D1 DBMS and its interface boundaries to the system. This can be done by stepping both calls to the function and its execution sequence, instruction by instruction. Assume, for this example, this procedure is used to identify the set of interfaces for the merge software module that are illustrated in Figure 7-4.

The merge software module requires two ordered ascending input lists. The location of each of these lists is identified as input to the merge function. In addition, the merge function requires the size of each input list and the location of the merged output list. Finally, a merge operation type is required along with a context specification. This context specification is used to identify if an input list entry is in correct context. If an entry is not in the required context it



MERGE SOFTWARE MODULE INTERFACES

FIGURE 7-4

is rejected and the next entry is selected. The merge software produces an output list that represents the merge of the two input lists for either an AND or OR merge operation type.

Also, once called, the merge software routine does not make external calls to other DBMS functions. Both input lists to be merged are brought into a local fast access memory where they are identified to the merge software routine. This assumption is made so the speed of a secondary memory access for retrievals of the input lists is not included in analyzing the performance improvement of the actual merging operation. The primary objective of this demonstration is to illustrate how the facility can be used to study details about choosing DBMS functions and evaluating SPFA's which can perform the function.

Once the merge software module's interface boundaries have been established, the facility provides the capability to dynamically test, in detail, the actual execution path of the merge module. This is conducted by actually following the execution path of the module using the facility. At this level, the facility permits execution of the modules instructions at either the source or machine level. Groups of instructions can be executed to help identify specific data dependencies of the merge routine. For instance, inefficient loops and/or dominate execution paths may be identified. If a dominate execution path is found, one approach, at this point, may be to re-write the routine to improve its performance as a software module.

Next, other types of data can be collected using the facility to help establish the impact on the quality of the DBMS by moving a software function into hardware. In this case, the facility can be used to help compute metrics for specific factors about individual DBMS software modules.

An example of a metric for the processing efficiency of DBMS software module can be computed from the following formula:

$M_{PE_{SW}}$ = metric for processing efficiency of a software module.

$$M_{PE_{SW}} = \sum_{i=1}^{9} X_i$$

Where X_1 = 1- $\frac{\text{\# non loop dependent statements in loop}}{\text{Total \# statements}}$

X_2 = Optimizing Compilier - 1- Yes;0 - No

X_3 = 1- $\frac{\text{\# compound exp defined more than once}}{\text{compound expressions}}$

"

"

"

"

X_8 =

X_9 = $\frac{\text{OS Linkage Time}}{\text{execution time}}$

The complete computation of this metric is part of the technical report (MCCA79).

The facility can be helpful in computing this metric because part of the computation requires examination of loops during execution of the module. The state of the module can be examined at specific points to identify any non-loop dependent variables. Once this metric has been determined, metrics for other factors as reliability or portability can be computed using the same procedure.

Once a set of metrics have been computed a user has an additional set of data from which to choose qualified candidates to be moved from software into hardware. Thus, the utilization data, interface data, dynamic testing data and quality data can then be reviewed to determine if overall application goals are enhanced by moving the DBMS function into hardware.

In this demonstration it is assumed that this data is used to choose the merge function. The merge function is a good candidate because it is a well defined DBMS function where a set of input/output boundary interfaces can be established. All inputs to this function are direct and once it begins execution it does not require an interface to other DBMS routines. Also, since we have already assumed the merge function is called excessively by the application programs from our example in Figure 7-3, its movement into hardware as a high performance SPFA machine may improve the overall performance of this application. Thus, this software module becomes a candidate to be replaced by a merge SPFA.

In summary, the facility can be utilized to establish candidate DBMS functions in several ways. First, it can be used to establish statistics on utilization and usage of the function. Next,specific interfaces to the function can be defined using the facility. In

addition, the facility permits testing the actual software function to help establish specific data dependencies it may have for the application it is being utilized for. Finally, the facility can be used to evaluate if specific factors regarding the software function can be improved by moving the function into hardware. Once a candidate is chosen, the next step is to develop SPFA's for the function.

7.3 INITIATE DEVELOPMENT OF MERGE SPFA'S USING FACILITY

The facility can be utilized to examine one or more SPFA's that may provide new architecture approaches for this function. Several SPFA's exist as alternative approaches to providing a merge capability in hardware. Some of the candidates include Batcher's odd-even merge network (BATC68), Stellhorn's inverted file processor (STEL77), which utilizes Batcher's odd-even network, and Hollaar's merge processor (HOLL76). It has been shown that from 10 to 100 times improvement may be gained using these SPFA's. These processors utilize different algorithms to merge input lists. The facility can be utilized to examine each as merge SPFA's and establish the advantages and disadvantages of each for this specific application. If neither can be used, the facility may also be used to either examine other SPFA's or tailor a SPFA for an application. This procedure requires use of the facility to create, test, and evaluate the alternative merge SPFA's. This development is now described and demonstrated for both Stellhorn's and Hollaar's merge processor.

7.3.1 Create Hollaar's Merge Processor

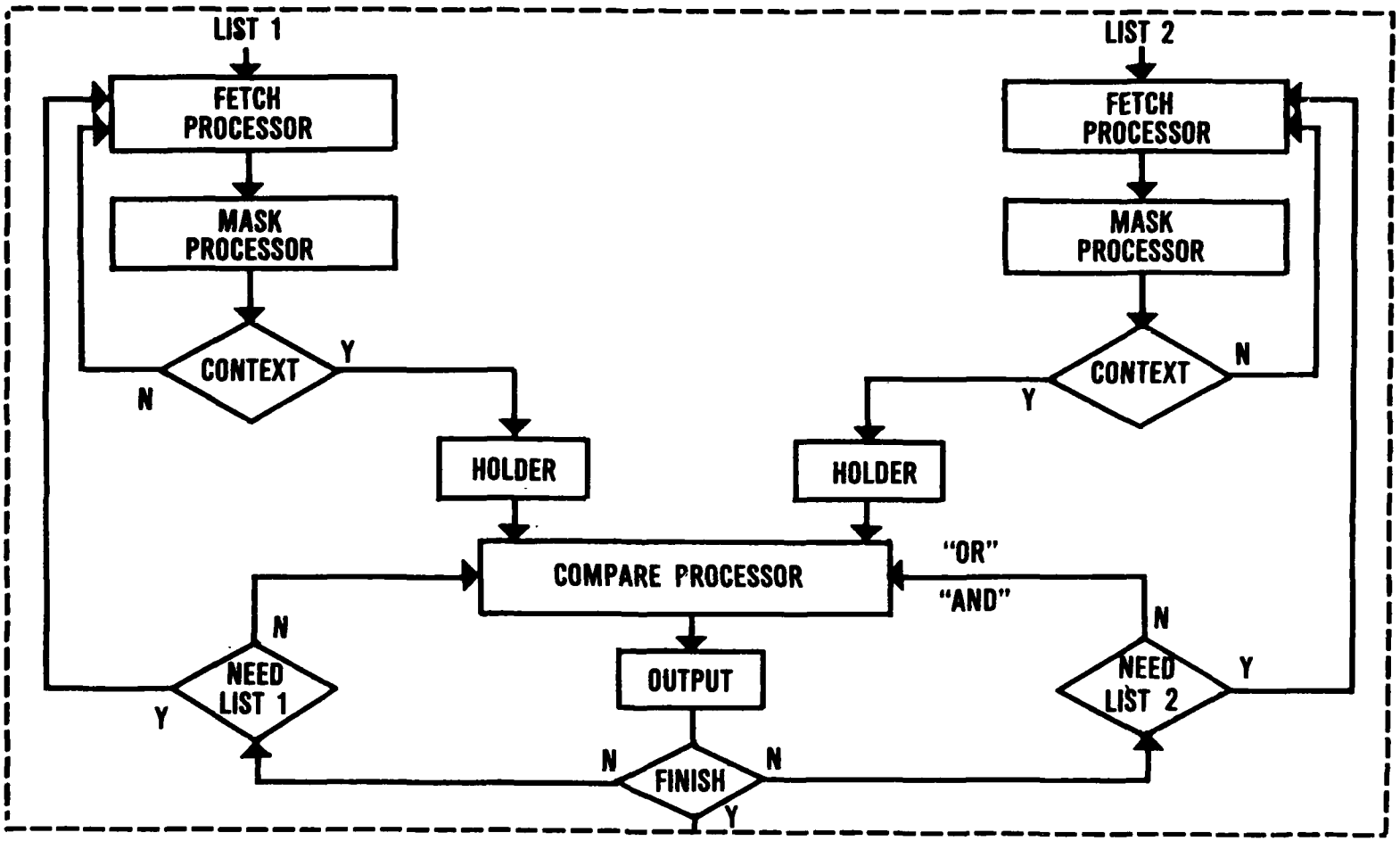
Hollaar's merge processor was created as a "MERG1" SPFA. A discussion of this procedure and a description of this SPFA in SMITE is

contained in Appendix A. The "MERG1" SPFA is illustrated in Figure 7-5 and consists of a fetch, mask, compare and output processor. The fetch processor fetches input entries, if needed in parallel, from list 1 and list 2. These entries are passed to the mask processor. The mask processor examines the entries from each list and determines if the entry is in proper context. If the entry is in correct context, it is placed in that list's holding register. If an entry is not in correct context, control returns to the fetch processor and the next entry is retrieved. This process continues until the holding register for both lists is full. When full, these entries are used as input to the compare processor. The compare processor then merges the entries for the merge type requested according to the merge algorithm in Figure 7-6.

This algorithm basically compares the two entries. If an OR operation is requested, an output is generated from the lower entry if the entries are unequal or either entry is equal. If an ADD operation is requested, an output is generated only if both entries are the same. In both cases, the list from where the output entry is taken is then marked empty and the fetch processor is begun. The processor continues to process entries from both lists until one is exhausted. When one of the lists completes the "MERG1" SPFA terminates.

7.3.2 Create Stellhorn's Inverted File Processor

Stellhorn's inverted file processor was used as the basis for creating a "MERG2" SPFA. The "MERG2" SPFA is illustrated in Figure 7-7 and a description of the SPFA, in SMITE, is contained in Appendix B. This SPFA consists of a fetch, mask, compare, coordination and output processors. The fetch processor fetches input from list 1 and list 2.



HOLLAAR'S MERGE PROCESSOR AS "MERGI" SPFA

FIGURE 7-5

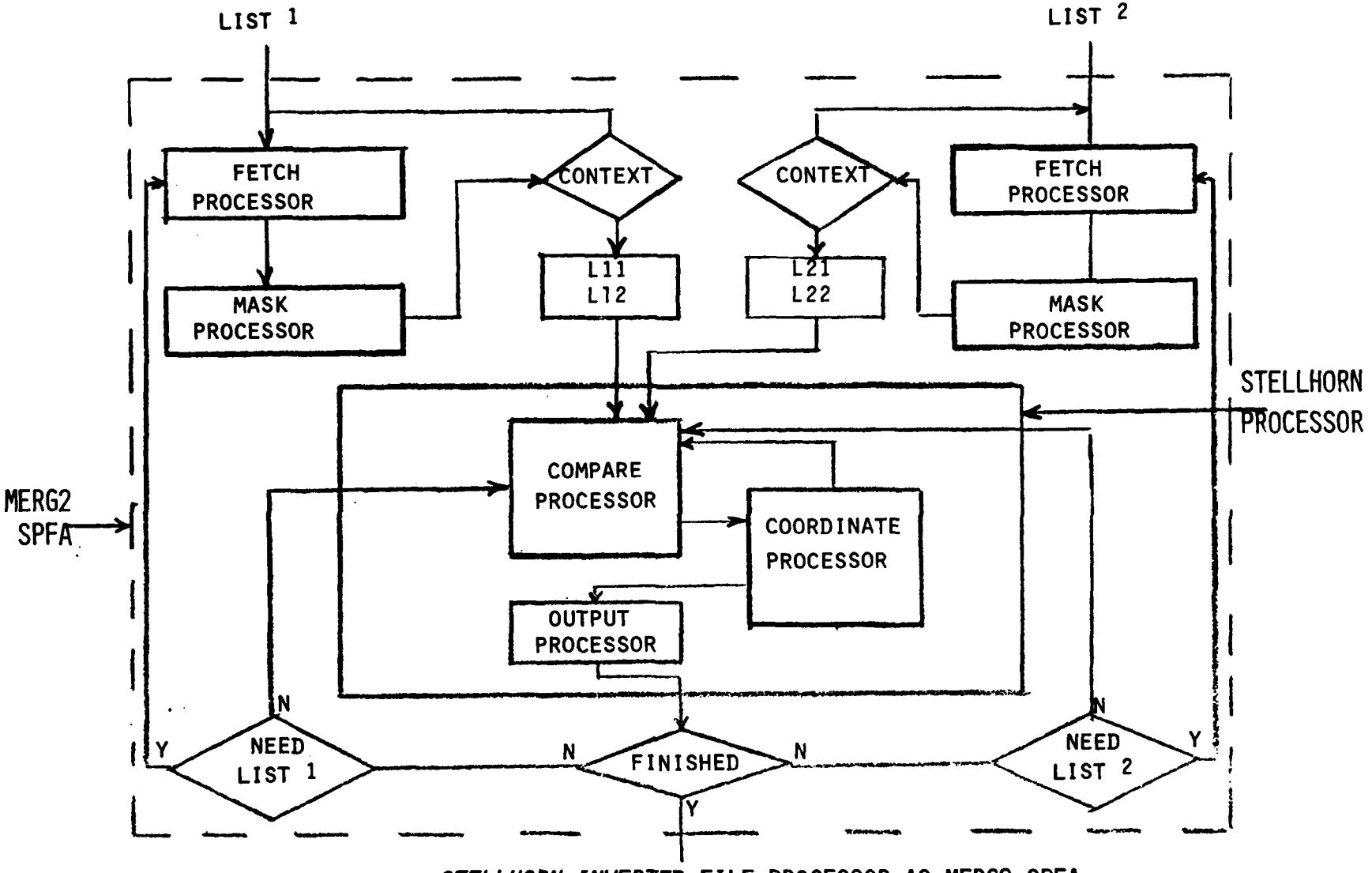
LIST 1 : LIST 2	OPERATION	ACTION
LESS THAN	AND	FETCH NEXT ENTRY FROM LIST 1
LESS THAN	OR	OUTPUT LIST = LIST 1 FETCH NEXT ENTRY FROM LIST 1
EQUAL	OR AND	OUTPUT LIST = LIST 1 OR LIST 2 FETCH NEXT ENTRY FROM LIST 1 FETCH NEXT ENTRY FROM LIST 2
GREATER THAN	OR	OUTPUT LIST = LIST 2 FETCH ENTRY LIST 2
GREATER THAN	AND	FETCH NEXT ENTRY FROM LIST 2

"MERGE1" SPFA OPERATIONS FOR COMPARE PROCESSOR

FIGURE 7-6

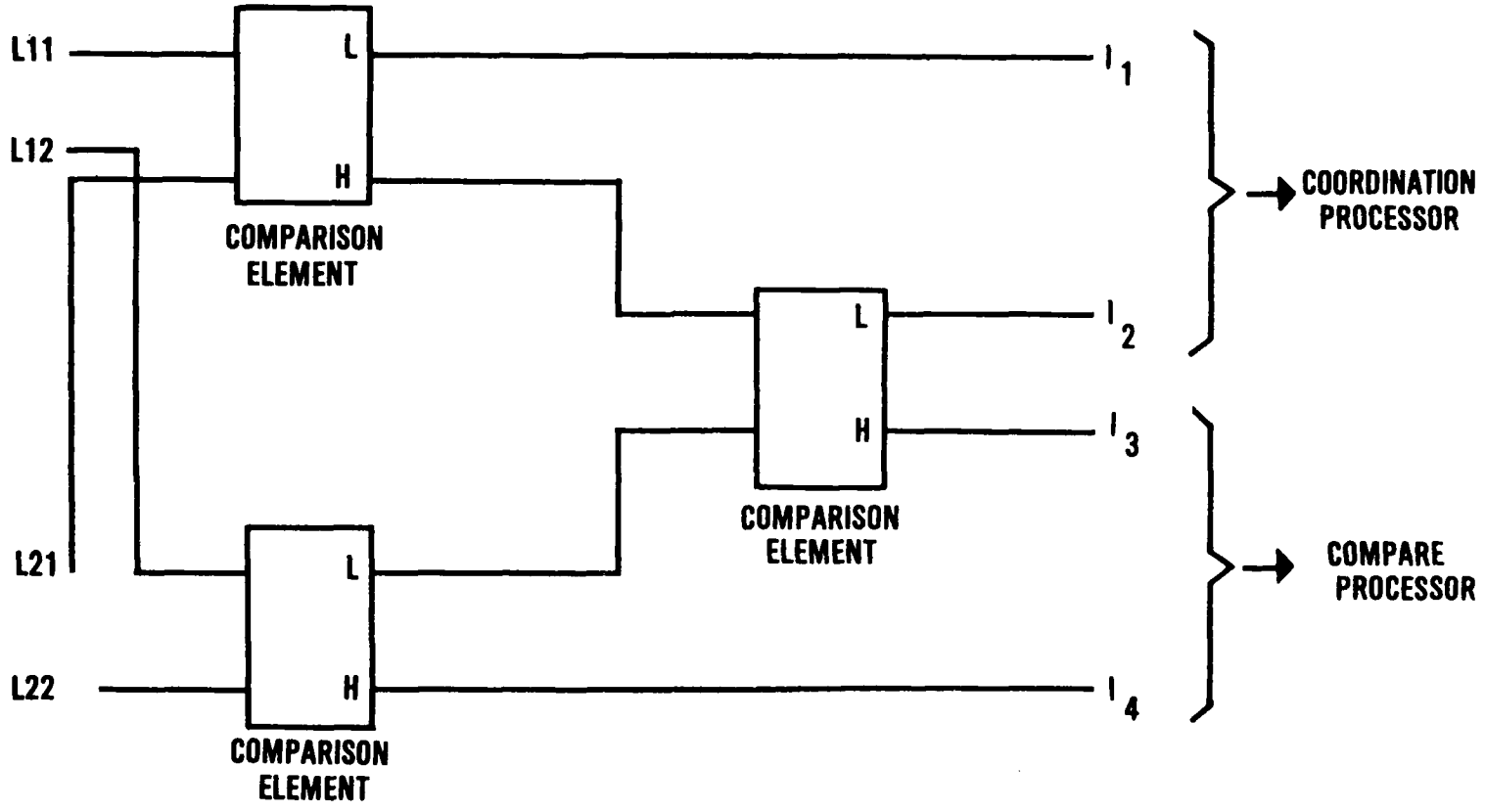
These entries are passed to the mask processor, The mask processor examines the entries from each list to determine if in proper context. This processor then presents the input entries in a parallel format to the compare processor. The compare processor merges the entries using an algorithm developed by Batcher. This algorithm is illustrated by the odd-even merge network illustrated in Figure 7-8. The basic element of the network is the comparison element. This element receives two inputs, performs an operation and presents their minimum on its L output and their maximum on its H output. A merge output consists of formulating these comparison elements into an odd merge network and an even merge network. In this manner, the coordination between the two arranges two ascendingly-ordered list of numbers into one ascendingly-ordered output list. The algorithm presents the odd indexed numbers of the two input lists to odd merge network while the even indexed numbers are presented to the even merge network. The lowest output of the odd network becomes the lowest number in the output list. The ith output of the even merge is compared with the i + 1 th output of the odd merge to form the 2 ith + 1 th numbers of the final list. If an output remains it becomes the highest number of the final list. A complete proof of this algorithm is given by Batcher (BATC68).

In order to illustrate development of this merge machine, a 2 X 2 merge network has been described for the "MERG2" SPFA compare processor using SMITE. The entries to the "MERG2" compare processor, illustrated in Figure 7-7, must be blocked into pairs and supplied as parallel sets of input. These entries, labeled L11 and L21 for list 1 entries and L21 and L22 for list 2 entries, are then merged as shown in Figure 7-8.



STELLHORN INVERTED FILE PROCESSOR AS MERG2 SPFA

FIGURE 7-7



2x2 MERGE NETWORK USED FOR COMPARE PROCESSOR FOR MERGE2 SPFA

FIGURE 7-8

L11 and L21 are compared in the odd network while L21 and L22 are compared in the even network. The compare processor produces an ascendingly ordered list from these sets of input.

The parallelism of the compare processor is illustrated below with the description of this processor in SMITE for the algorithm illustrated in Figure 7-8.

```
COMPARE:   PROCESSOR;
PARALLEL-BEGIN;
   COMP (L11, L21);
   COMP (L12, L22);
PARALLEL-END;
   COMP (L12, L21);
COMPARE:   END;
```

The odd indexed entries of the merge network are compared as L11 and L21 and even indexed entries are compared as L12 and L22. COMP is the comparison element called that identifies the lower input entry and the higher input entry. When the parallel compare is completed, the highest output from the odd indexed network (L12) is compared to the lowest output of the even indexed network (L21). The lower pair is then presented to the coordination processor and the upper pair is recycled back to the compare processor. New entries are then fetched while the compare and coordination processor execute.

The coordination processor is needed because the odd-even merge network algorithm used by the compare processor does not remove duplicate output entries. The coordination processor is used to form the OR and AND of the two lists. Duplicate entries are removed for the OR operation and single entries are removed for the AND operation. When complete, the coordination processors transfers entries to the output list. When an input list finishes the "MERG2" SPFA terminates if the request is for an AND operation. If the request is for an OR operation the remaining entries in the unfinished list are processed as outputs.

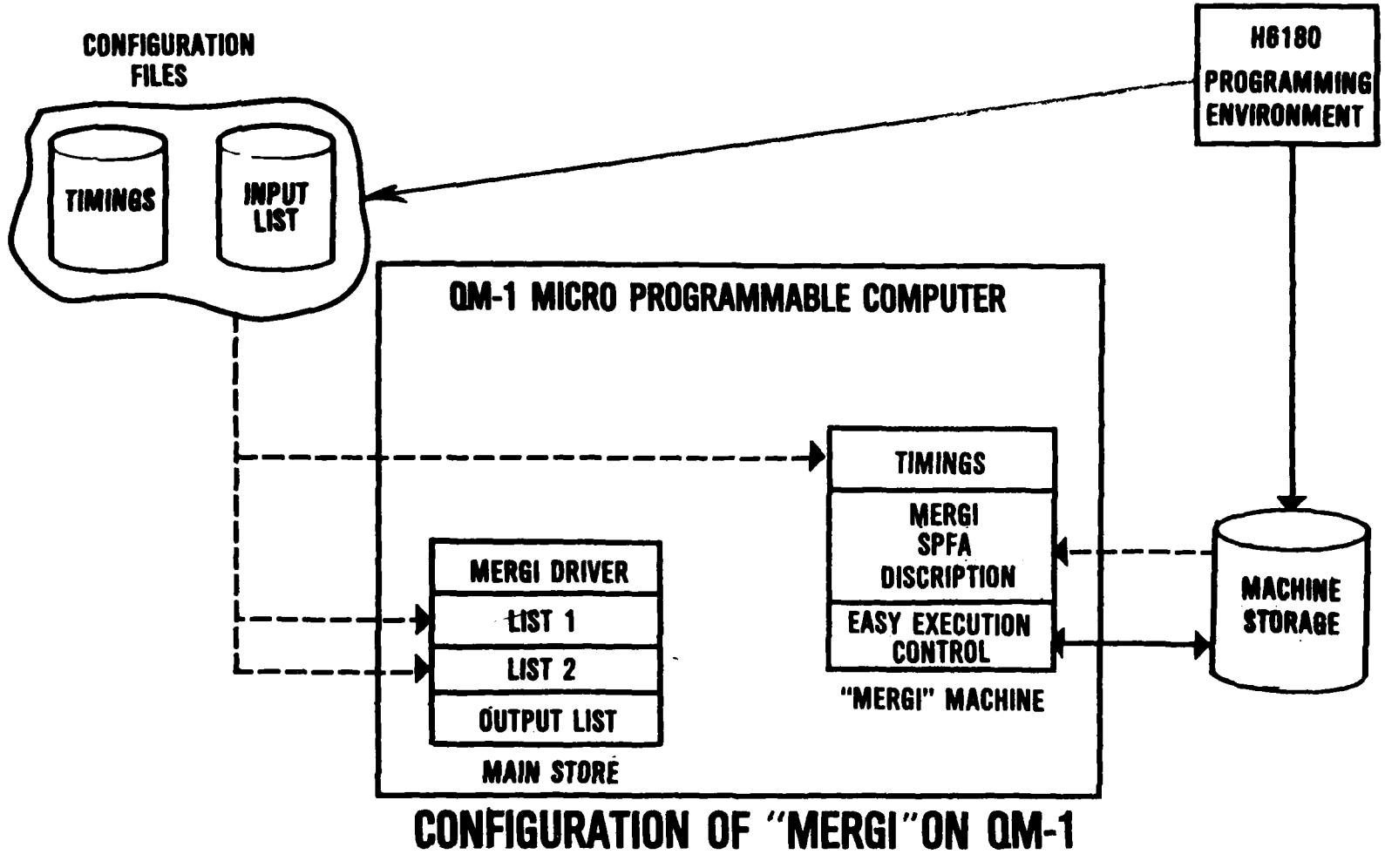
7.4 EXECUTION OF THE MERGE MACHINES IN THE FACILITY

Each merge machine was configured in the facility on the QM-1. Illustrated in Figure 7-9 is the configuration of the "MERG1" SPFA machine. The description of the "MERG1" SPFA is transferred from a local machine storage to control store on the QM-1 where the actual configuration of the "MERG1" machine is performed by the following procedure.

A set of configuration files were established on the QM-1 that consist of various drivers, input lists and timing assignments. Examples of these file types are illustrated in Figure 7-10. These files were created using the EDIT and Execute commands available from the EASY execution control module. For each merge machine one or more configuration files can be selected. The selection of these files establishes various driver, input lists and timing assignments for each machine. These files, when selected, help establish the desired configuration without manually having to reproduce the configuration for each machine.

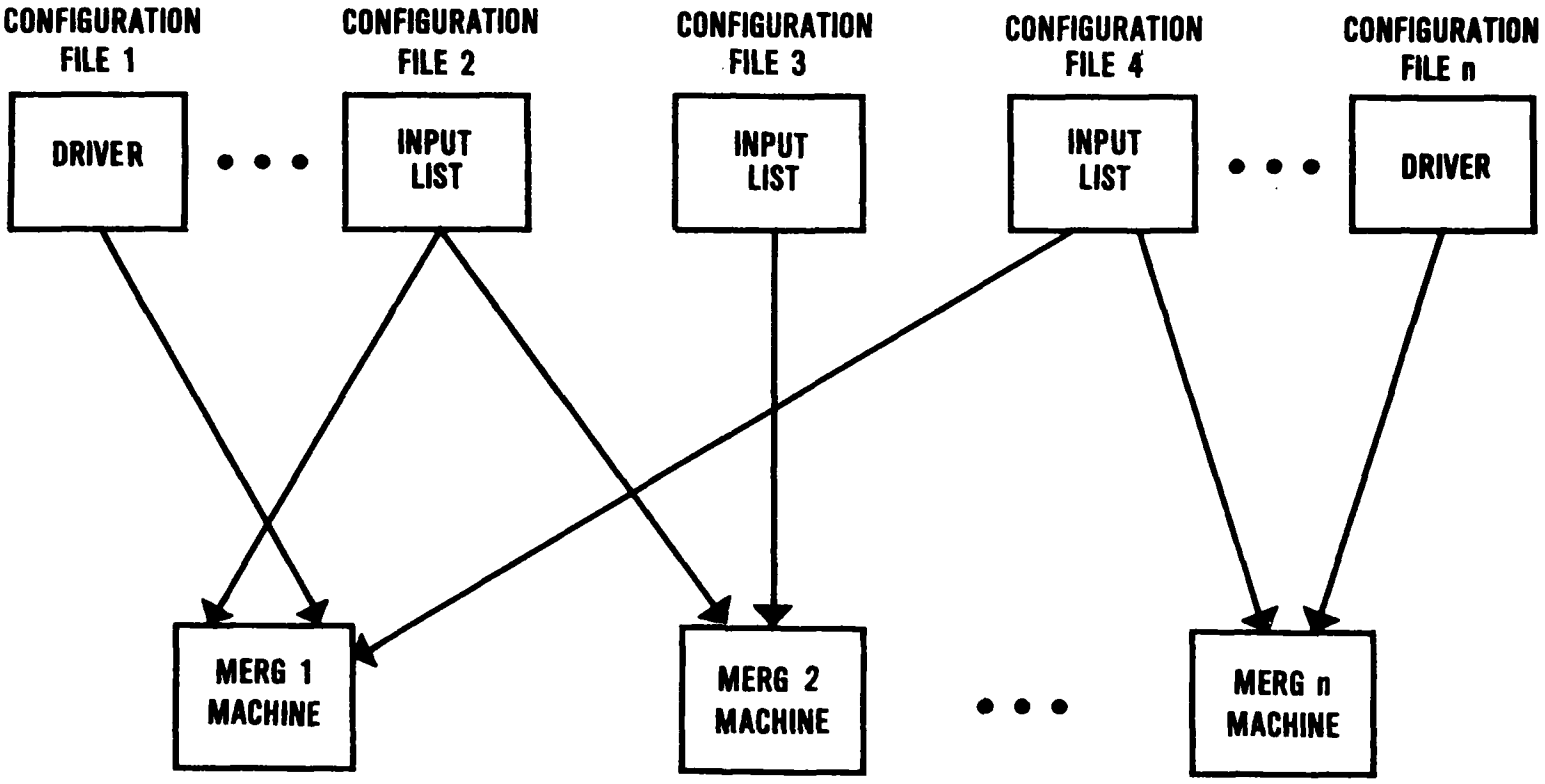
This technique has been especially important for evaluating the merge machines. It enables configurations to be set up and changed easily. It also enables similar conditions to be established for different SPFA's machines. It also enables many combinations of drivers and input lists to be presented to each machine without manually inputing the items. Such a capability is needed in a DMAD Facility and should be provided as an automated tool.

Part of the configuration of each machine requires a driver to instruct the execution of the machine. This driver must identify the same interfaces that were established during the select candidate



CONFIGURATION OF "MERGI" ON QM-1

FIGURE 7-9



USE OF CONFIGURATION FILES FOR MERGE MACHINES

FIGURE 7-10

function process to insure compatibility between the SPFA and software module it is replacing on the DBMS machine.

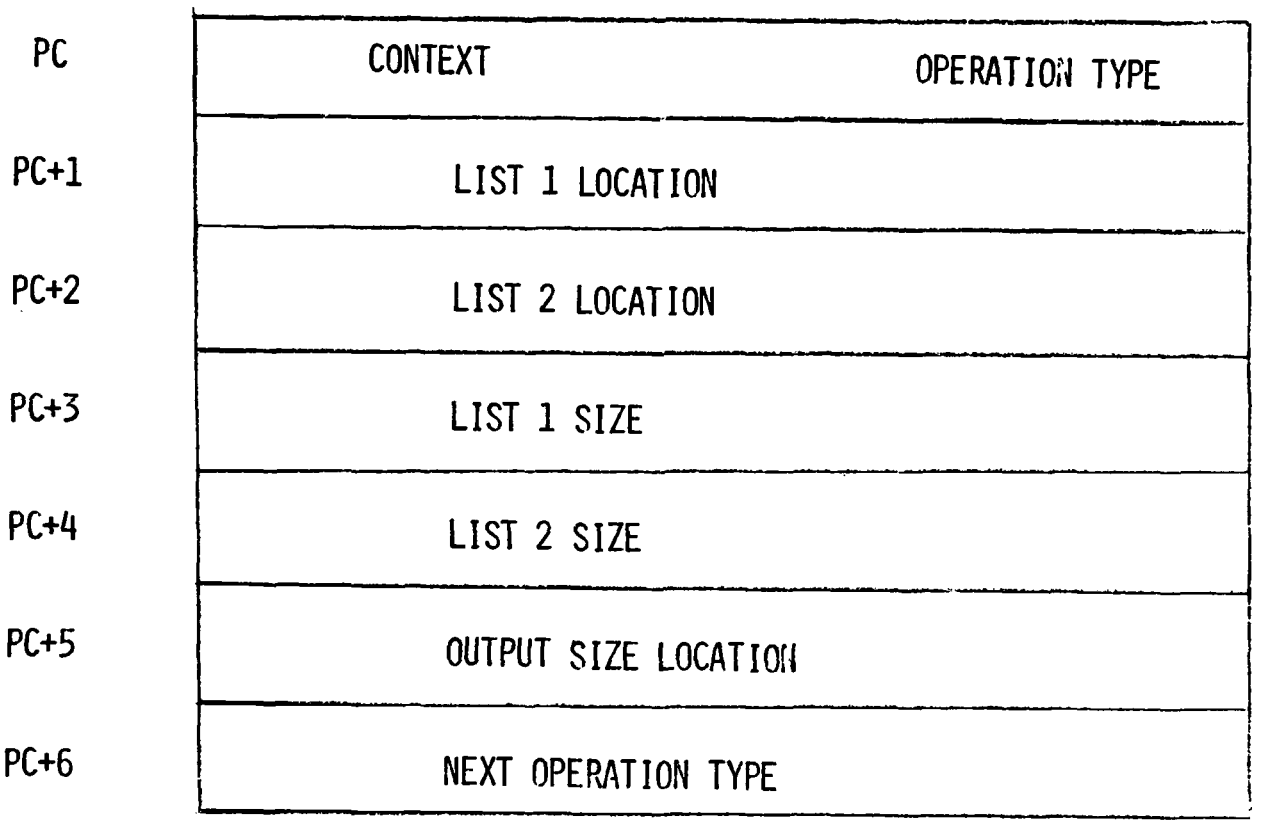
An illustration of a driver for the "MERG1" SPFA is given in Figure 7-11. The driver defines the merge operation type, input context, list 1 and list 2 location, list 1 and list 2 sizes and output list location.

For this demonstration the "MERG1" SPFA can perform two merge operation types:

Operation Type	
0	- AND input lists
1	- OR input lists

Other merge request types are possible (HOLL79) and can be added to both merge machines. The input driver also identifies a context for input list entry. If an entry is not in correct context, it is rejected by the merge machine and the next entry from the same list is retrieved. The two input lists, list 1 and list 2, are located in a configuration file.

Once the "MERG1" SPFA is fully configured on the QM-1 it becomes a "MERG1" machine. At this point, execution can begin. Figure 7-12 illustrates several portions of the "MERG1" machine on the QM-1 that can be examined during execution. In this figure, portions of main store are illustrated where the input data lists are contained. The values in control store are the timings assigned to the "MERG1" machine prior to an execution. A timing control clock for the merge machines is provided as part of the EASY/SMITE/SASS support package on the QM-1 (TRW79). Illustrated in Figure 7-12 are the local storage registers (LS24 and LS25) that are used as the timing control clock and accumulate the timing data for the merge machines.



SAMPLE INPUT DRIVER FOR MERG1 SPFA

FIGURE 7-11

```
!PRINTCS,0,200,0.                                    UNLOCKED    1.1.2

000000:  000000  000006  000000  000000  000000  000011  000000  000011
000010:  000000  000000  000000  000017  000000  000017  000000  000015
000020:  000000  000015  000000  000017  000000  000015  000000  000017
000030:  000000  000017  000000  000017  000000  000017  000000  106400
000040:  000000  000143  000000  000302  000000  020003  000000  020004
000050:  000000  000000  000000  000003  000000  000502  000000  000035
000060:  000000  000036  000000  020005  000000  020004  000000  000000
000070:  ALL ZEROS
000100:  000000  020005  000000  020004  000000  000021  002210  000000
000110:  002212  000000  002242  000000  000000  000000  000000  000000
000120:  ALL ZEROS
000130:  ALL ZEROS
000140:  ALL ZEROS
000150:  ALL ZEROS
000160:  ALL ZEROS
000170:  ALL ZEROS
000200:  ALL ZEROS
000210:  ALL ZEROS
000220:  ALL ZEROS
000230:  ALL ZEROS
000240:  ALL ZEROS
000250:  ALL ZEROS
000260:  ALL ZEROS
000270:  ALL ZEROS
000030 K  000000  000017  000000  000017  000000  000017  000000  106400
000040 K  000000  000143  000000  000302  000000  020003  000000  020004
000050 K  000000  000000  000000  000003  000000  000502  000000  000035
000060 K  000000  000036  000000  020005  000000  020004  000000  000000
000070 K  000000  000000  000000  000000  000000  000000  000000  000000
!REGISTERS,ADR=20000,TITLE FMT = . X
LOCAL STORE
    LS00    LS01    LS02    LS03    LS04    LS05    LS06    LS07
  000000  000000  000000  000000  000000  000000  000000  000000
    LS10    LS11    LS12    LS13    LS14    LS15    LS16    LS17
  000000  000000  000000  000000  000000  000000  000000  000000
    LS20    LS21    LS22    LS23    LS24    LS25    LS26   R.SYS
  000004  000001  000000  020000  000000  000110  020000  000000
   R.MX    R.IX    R.IY   R.MPC   R.ADR   FIDX    FMPC    FIST
  020001  000013  000000  002162  000000      20      33      00
```

REALIZATION TIMING

CONTROL ARRAY ENTRIES

(CONTROL STORE)

TIMING CONTROL

ILLUSTRATION OF MERGE MACHINE
ON QM-1 MICROPROGRAMMABLE COMPUTER
FIGURE 7-12

Both merge machines were tested on the QM-1 prior to evaluation. The testing process utilized the Debugging Model (FINF79) to identify description errors in both machines. The facility was fully capable of establishing and verifying errors utilizing this model. The debugging tools provided enabled the testing process to be completed successfully. When testing was complete, the evaluation process was conducted.

In summary, the development of both merge machines, to this point, illustrates how a facility can be used to describe and configure several SPFA machines that perform the same DBMS function. In this development various types of information have already been collected. For instance, each of the merge machines considered utilizes a different algorithm to merge input lists.

In this example the "MERG1" SPFA compare processor merges the inputs serially while the algorithm for the "MERG2" SPFA compare processor merges the inputs in parallel. The facility can be used to study advantages and disadvantages of each approach. The documentation of each machine, in SMITE, including individual architectural constructs, serves as important information source which can be analyzed during tradeoff analysis of the various machines. Described in the next section is a description of the procedure used to gather timing data during execution of the merge machines.

7.5 USE OF FACILITY TO EVALUATE MERGE MACHINES DATA

Illustrated in Figure 7-13 are the individual processors that comprise the "MERG1" and "MERG2" machines. For each processor a sequence of operations is listed. These are operations required by each processor to complete its function. These operations are described by a sequence of one or more SMITE statements for each machine.

A select and transfer operation are required by the fetch processor

PROCESSOR / MACHINE	FETCH	MASK	COMPARE	COORDINATION	OUTPUT/ RECYCLE
MERG1 MACHINE	SELECT TRANSFER	SELECT COMPARE TRANSFER	SELECT COMPARE TRANSFER OR INDICATOR SET	N/A	TRANSFER
MERG2 MACHINE	SELECT TRANSFER	SELECT COMPARE TRANSFER SELECT INDICATOR SET	COMPARISON	TRANSFER SELECT COMPARE TRANSFER	TRANSFER

MERGE SPFA MACHINE OPERATIONS

FIGURE 7-13

for both the "MERG1" and "MERG2" SPFA. The mask processor for the "MERG1" machine examines the context of each input entry and then transfers acceptable entries to a holding register and rejects entries not in correct context. The mask processor for the "MERG2" SPFA examines the context of input entries and then presents a set of entries in parallel to the compare processor. The compare processor for the "MERG1" SPFA performs a select, compare and transfer operation or a select, compare and indicator set operation depending on merge type. The compare processor for the "MERG2" SPFA is described as a 2 X 2 merge network. For this network three comparison elements are needed with two elements executing in parallel. The "MERG2" coordination processor accepts two input entries and removes duplicates for an OR merge type or selects duplicates for an AND merge type. The operations required include transfer the input entries, select operation to identify merge type, compare operation to identify single or duplicate entries and transfer of selected output entries.

If the SPFA is actually implemented in hardware the operations of each processor can be performed by selected hardware components. Standard hardware components can be used for such operations as a compare or select. However, specialized logic circuits may be needed to perform operations needed by the comparison element for the "MERG2" SPFA. For example, Batcher proposes such a circuit for this element (BATC68).

Several strategies for implementing each machine in hardware can be examined by using the facility to assign times to complete each operation based on the time of a selected components or computing time from a specialized logic design. Both approaches were used for the merge

machines and timings were assigned to each machine's individual processors as follows:

Let time for fetch processor = $t_{(0,m)}$

time for mask processor = $t_{(3,m)}$

and time for compare processor = $t_{(6,m)}$ "AND" merge type

$t_{(10,m)}$ "OR" merge type

The time $t_{(0,m)}$ is the time defined as TIMER-VALUE (0) in the "MERG1" description for the fetch processor. The time assigned to TIMER-VALUE (0) during the execution of the "MERG1" machine is either $t_{(1,m)}$ or $t_{(2,m)}$. These times are defined as TIMER-VALUE (1) and TIMER-VALUE (2). If a fetch is made from list 1 then TIMER-VALUE (1) is assigned to TIMER-VALUE (0). If a fetch is made only from list 2 then TIMER-VALUE (2) is assigned to TIMER-VALUE (0). If a fetch is made from both list 1 and list 2 in parallel then the maximum time is assigned to TIMER-VALUE (0). The time $t_{(3,m)}$ for the mask processor is assigned similarly. Time $t_{(3,m)}$ is determined from $t_{(4,m)}$ or $t_{(5,m)}$.

The time for the compare processor is based on both the merge type requested and the criteria of the actual merge. For instance, the OR merge type always produces an output entry while the AND merge type may not. The "MERG1" compare processor requires the operations that complete the algorithm in Figure 7-6. For instance, the time for an "AND" merge type $t_{(2,m)}$ is based on whether the list entries to be compared are either less than, equal or greater than. The time is assigned accordingly as $t_{(3,m)}$ OR $t_{(4,m)}$ or $t_{(5,m)}$. Similarly, the

time for the OR merge type is based on the action resulting from the comparison of the inputs. This time is either $t_{(7,m)}$ or $t_{(8,m)}$ or $t_{(9,m)}$. The actual time assigned to the machine for the compare processor is accumulated during execution.

A set of times were similarly, assigned to the "MERG2" SPFA processor and are divided into:

Let time for fetch processor = $t_{(0,m)}$

time for mask processor = $t_{(3,m)}$

time for compare processor = $t_{(6,m)}$

time for coordination processor = $t_{(8,m)}$ "AND" merge type, $t_{(11,m)}$ "OR" merge type

time for output (l_1) processor = $t_{(17,m)}$

and time for output (l_2) processor = $t_{(18,m)}$

7.5.1 Sample Timing Calculations

The actual times assigned to each of the processors for both machines was based on assignment from each machines' realization timing control array. Times were computed for the "MERG1" RTCA for two realization assumptions. Realization assumption one is based on implementing operations using Shottkey logic while realization assumption two is based on MECL logic. These technology types were selected to illustrate

a capability of the facility to help examine tradeoffs between the various types available. For instance, as illustrated in Table 5-1, gate delay time for Shottkey logic is 3 nanoseconds (ns) while the gate delay time for MECL logic is 2 ns. However, the Shottkey logic costs less with lower power/gate.

The timings for the operations illustrated in Figure 7-14 are based on manufacturers' stated times from references(LANC77,SCHT77,MECL74). For example, for a compare operation the times are based on selected components such as MC10166 (MECL comparator) or AM25652521 (TTL comparator). The time for the comparison operation is based on Batcher's special design that requires 14 gate delays. These times were used to compute entries in the realization assumption timing control array for each machine.

A set of sample calculations for the "MERG1" RTCA, using realization assumption one, are described below;

Fetch Processor (TIMER-VALUE (0))

$$t_{(0,1)} = \max(t_{(1,m)} \text{ or } t_{(2,m)})$$

$$t_{(0,1)} = \max(t_{select} + t_{trans}) = 23.5 \text{ ns}$$

Compare Processor (TIMER-VALUE (6))

$$t_{(6,1)} = t_{(7,1)} \text{ or } t_{(8,1)} \text{ or } t_{(9,1)}$$

$$t_{(7,1)} = t_{select} + t_{compare} + t_{indicator\ set} = 35.5 \text{ ns}$$

$$t_{(8,1)} = t_{select} + t_{compare} + t_{trans} = 35.5 \text{ ns}$$

$$t_{(9,1)} = t_{select} + t_{compare} + t_{indicator\ set} = 35.5 \text{ ns}$$

OPERATIONS	REALIZATION ASSUMPTION 1 (SHOTTKEY) (NS)	REALIZATION ASSUMPTION 2 (MECL) (NS)
SELECT $_{(t\ select)}$	6	3
COMPARE $_{(t\ compare)}$	15	6
TRANSFER $_{(t\ trans)}$	11.5	6
INDICATOR SET $_{(t\ ind\ set)}$	11.5	4
SHIFT $_{(t\ shift)}$	19	3.3
COMPARISON	42	28

TIMES FOR OPERATIONS REQUIRED BY SPFA MACHINES

FIGURE 7-14

The completed realization timing control array for the "MERG1" SPFA is illustrated in Figure 7-15 for both realization assumption one and two. Next, a similar set of calculations were performed for the "MERG2". The times entered in this RTCA, illustrated in Figure 7-16 are based on realization assumption two. Once the RTCA's were generated, several tests were conducted to illustrate how the facility can be used to generate timing for the two merge SPFA's.

In summary, a set of timings are assigned to each machine. These times are assigned based on required operations of each machine identified from its description. These times are based on utilizing different hardware characteristics to complete the operation. Once timings are assigned, the facility is used to evaluate the architecture of each machine during execution in the facility and generate timing data during actual merge requests.

OPERATIONS (TIMER-VALUES)	MERG 1 SPFA	
	RA 1 NS	RA 2 NS
0 (FETCH)	-	-
1	17.5	9
2	17.5	9
3 (MASK)	-	-
4	32.5	15
5	32.5	15
6 (COMPARE)	-	-
7 } AND	32.5	13
8 } AND	32.5	15
9 } AND	32.5	13
10 (COMPARE)	-	-
11 } OR	32.5	15
12 } OR	32.5	15
13 } OR	32.5	15
14 (OUTPUT)	11.5	6

$MERG_1$ REALIZATION TIMING CONTROL ARRAY

FIGURE 7-15

OPERATIONS (TIMER-VALUE)		MERG 2 SPFA RA 2 (NS)
0	(FETCH)	-
1		11
2		11
3	(MASK)	-
4		26
5		26
6	(COMPARE)	-
7	(COMPARISON)	34
8	(COORDINATE)	-
9	AND	17
10	AND	25
11	(COORDINATE)	-
12	OR	17
13	OR	25
14	OUTPUT/RECYCLE	6

MERG$_2$ REALIZATION TIMING CONTROL ARRAY

FIGURE 7-16

7.5.2 Evaluation Results

This section consists of several examples of how the facility can be used to evaluate the merge machines. These examples include:

(1) using the facility to study the basic architectural constructs of each machine by studying the data flow and generation of output for each machine,

(2) using the facility to generate timing data for each machine for different realization assumptions,

(3) use of facility to establish unique types of input data to study reactions of the merge machines based on their individual merge algorithms, and

(4) using the facility to modify and re-create an existing machine that is tailored to specific criteria.

Several types of facility capabilities are used to examine the flow of each machine during execution. Controls are set to examine the machines' state at the entrance and exit of each machine's processor. For example, the fetch processor retrieves data in parallel from each input list. The facility is used to examine this data flow at the initiation of this processor and at its completion to insure a proper data flow. This type of analysis was used to examine the concurrent execution of the retrieval of entries from the input lists.

7.5.2.1 MERG1 Machine

Next, several types of executions were made to illustrate using the facility to generate timing data. First, timing data was generated for the "MERG1" machine. A cycle of the machine includes fetch, mask, and compare operations. An OR merge type using assignments based on realization assumption one requires 82.5 ns for this cycle. This time

compares with the 80 ns computed for the merge processor (HOLL76) which was based on implementation in the same logic, However, the time for the merge processor was computed from an actual logic circuit of the processor while the timing for the merge machine is based on using the facility. Next, the "MERG1" machine was configured with the same input lists, the same driver, and with timing assignments from realization assumption two. A complete cycle of this machine for the same OR merge type request is illustrated in Figure 7-17 and requires 47_8 ns or 39 ns. This comparison is illustrated in Figure 7-18 where the timing generated for the same merge machine based on two different realization assumptions are shown.

This demonstration illustrates how the facility can be utilized to generate timing data from the "MERG1" machine based on timing values assigned using different hardware technologies. For instance, the data collected illustrates that the machine can perform faster with higher speed technology. At this point, additional analysis can be performed. For instance, the timing data generated is based on only one cycle of the machine.

The facility can be utilized to examine execution of the machine for large sets of input lists. An example of actual "MERG1" machine output for two input lists, each containing 50 entries, for an OR merge type is illustrated in Figures 7-19, 7-20, and 7-21. The time to perform this merge was accumulated directly on the QM-1 for 70 cycles and illustrated in Figure 7-21 as 4257_8 or 223 ns.

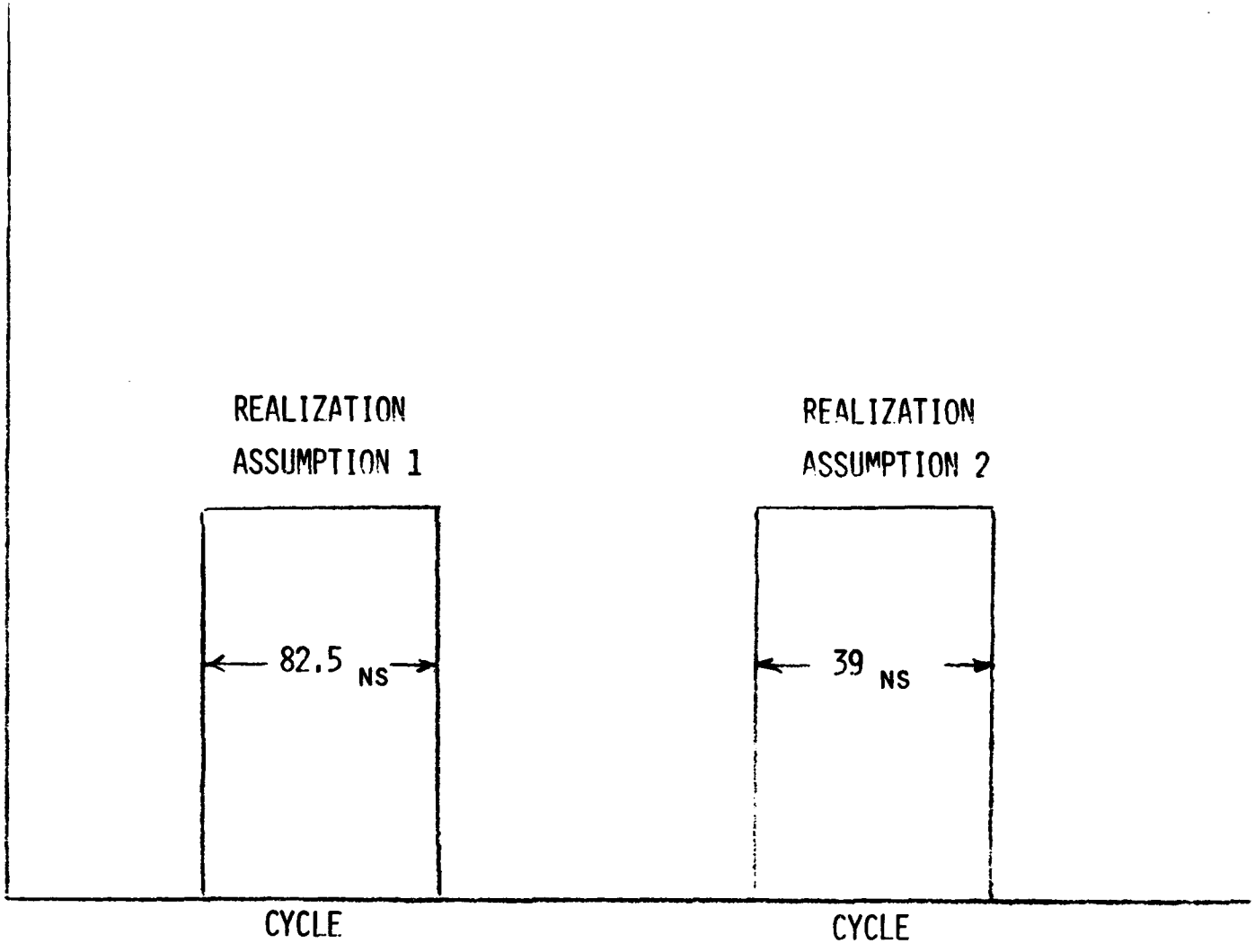
Similarly, other various types of input lists can also be configured at this point to generate additional timing data. The types of input lists can be larger or can be varied to consist of any unique type needed for a specific application. The "MERG1" machine can then be utilized to

```
<CR>. X
!<CR>. XISPLAY GOES HERE; STATE IS IDLE
!<CR>. XISPLAY GOES HERE; STATE IS STEP
!STATE DISPLAY GOES HERE; STATE IS STEP
   IN LINE 70 OF PROC/FUNC SKBCHA IN MODULE SKYBRD
!REGISTERS,ADR=20000,TITLE FMT = . X
LOCAL STORE
    LS00     LS01     LS02     LS03     LS04     LS05     LS06     LS07
  000000   000000   000000   000000   000000   000000   000000   000000
    LS10     LS11     LS12     LS13     LS14     LS15     LS16     LS17
  000000   000000   000000   000000   000000   000000   777777   777761
    LS20     LS21     LS22     LS23     LS24     LS25     LS26    R.SYS
  000004   000001   000021   020000   000000   000047   020000   000000
    R.MX     R.IX     R.IY    R.MPC    R.ADR     FIDX     FMPC     FIST
  020001   000013   000000   002154   000000       20       33       00
!<CR>. X
```

TIMING CONTROL CLOCK

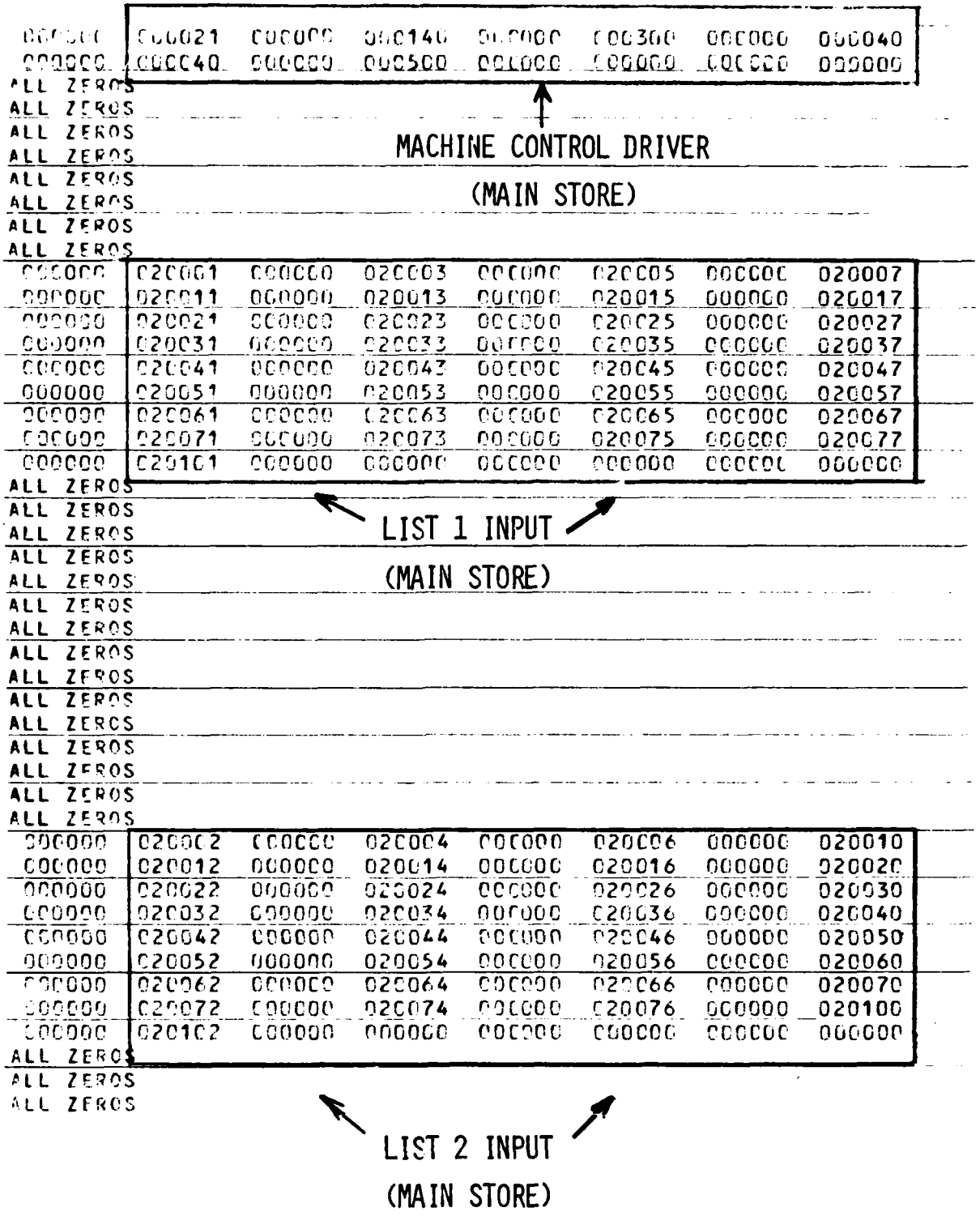
RESULTS OF ONE CYCLE OF MERG1 MACHINE
FOR REALIZATION ASSUMPTION 2

FIGURE 7-17



TIMING DATA FOR MERG 1 MACHINE

FIGURE 7-18



SAMPLE INPUT LISTS CONFIGURED FOR "MERG1" MACHINE

FIGURE 7-19

ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS

OUTPUT LIST

000000	000001	000000	000002	000000	000003	000000	000004
000000	000005	000000	000006	000000	000007	000000	000010
000000	000011	000000	000012	000000	000013	000000	000014
000000	000015	000000	000016	000000	000017	000000	000020
000000	000021	000000	000022	000000	000023	000000	000024
000000	000025	000000	000026	000000	000027	000000	000030
000000	000031	000000	000032	000000	000033	000000	000034
000000	000035	000000	000036	000000	000037	000000	000040
000000	000041	000000	000042	000000	000043	000000	000044
000000	000045	000000	000046	000000	000047	000000	000050
000000	000051	000000	000052	000000	000053	000000	000054
000000	000055	000000	000056	000000	000057	000000	000060
000000	000061	000000	000062	000000	000063	000000	000064
000000	000065	000000	000066	000000	000067	000000	000070
000000	000071	000000	000000	000000	000000	000000	000000

ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS
ALL ZEROS

TIMING ASSIGNMENTS

,0,200,0.

UNLOCKED

000000	000006	000000	000000	000000	000011	000000	000011
000000	000000	000000	000017	000000	000017	000000	000015
000000	000015	000000	000017	000000	000015	000000	000017
000000	000017	000000	000017	000000	000017	000000	002020
000000	000175	000000	000335	000000	020071	000000	020072
000000	000000	000000	000071	000000	000571	000000	000003

MERG1 MACHINE OUTPUT FOR 'OR' MERGE TYPE REQUEST

FIGURE 7-20

```
000220:  ALL ZEROS
000230:  ALL ZEROS
000240:  ALL ZEROS
000250:  ALL ZEROS
000260:  ALL ZEROS
000270:  ALL ZEROS
000060 K  000000  000003  000000  000000  000000  000000  000000  000000
000070 K  000000  000000  000000  000000  000000  000000  000000  000000
!REGISTERS,ADR=20000,TITLE FMT = . X
LOCAL STORE
     LS00    LS01    LS02    LS03    LS04    LS05    LS06    LS07
  000000  000000  000000  000000  000000  000000  000000  000000
     LS10    LS11    LS12    LS13    LS14    LS15    LS16    LS17
  000000  000000  000000  000000  000000  000000  777777  777761
     LS20    LS21    LS22    LS23    LS24    LS25    LS26   R.SYS
  000004  000001  000021  020000  000000  004257  020000  000000
     R.MX    R.IX    R.IY   R.MPC   R.ADR    FIDX    FMPC    FIST
  020001  000013  000000  002154  000000      20      33      00
!<a>. X
   IN LINE 70 OF PROC/FUNC SKRCHA IN MODULE SKYBRD
!<LF>. X
```

TIMING CONTROL CLOCK

TIMING RESULTS FOR MERG1 MACHINE FOR "OR" MERGE TYPE REQUEST

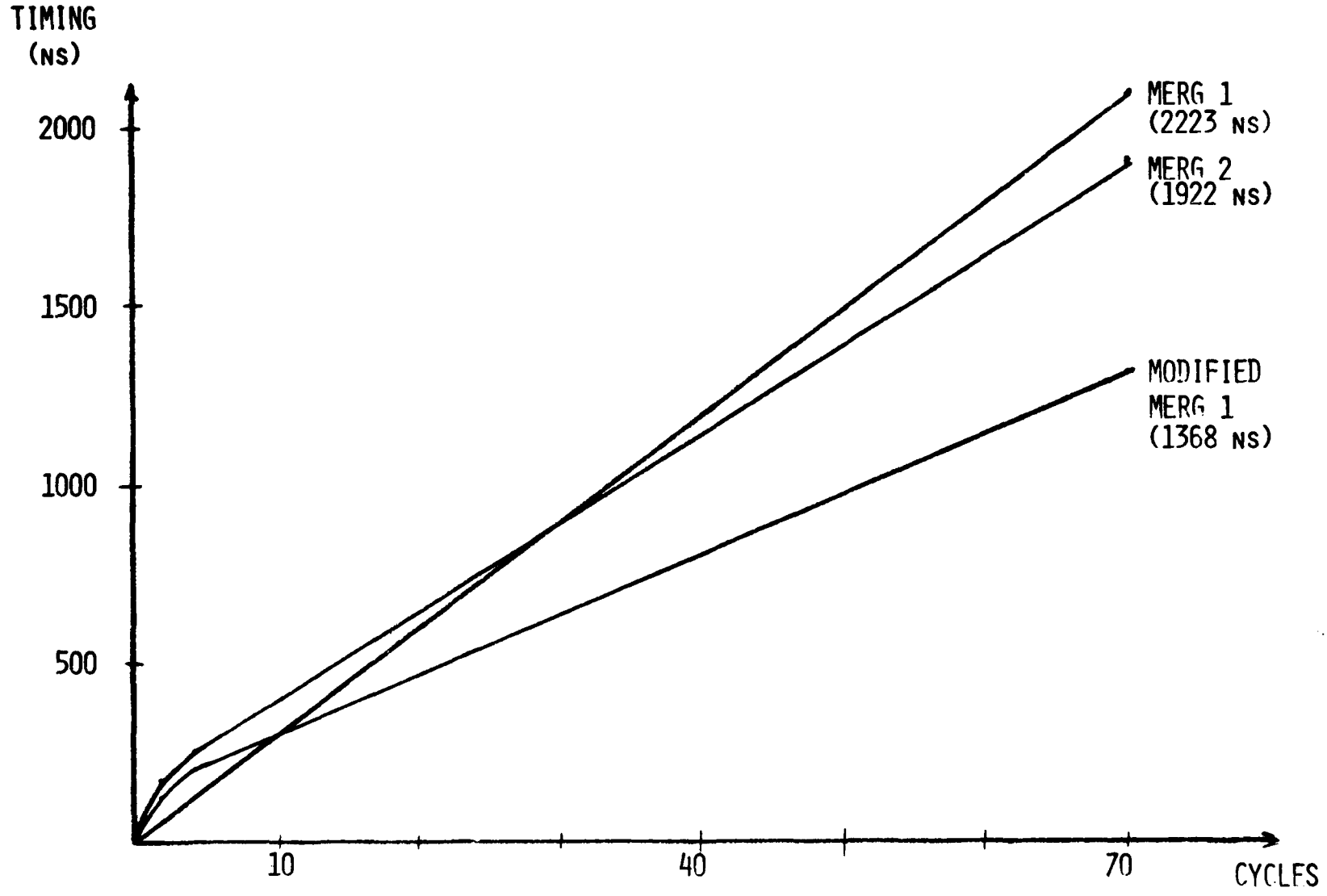
FIGURE 7-21

generate additional timing data or to study how the machine merges the lists.

Next, the facility can be used to study the actual architecture modifications of the "MERG1" SPFA. For instance, a user can modify a SPFA to determine if its performance can be improved for a specific application. In this example, the fetch, mask, and compare processors of the "MERG1" machine all execute serially. However, for the odd-even lists used a fetch is always made alternatively from either list 1 or list 2. Thus, the "MERG1" machine can be modified so its fetch and mask processor execute in parallel with its compare processor. A description of the modified "MERG1" machine is illustrated in Appendix C. This machine was also configured on the QM-1 with the same timing assignments from the RTCA, the same input lists and the same driver as in the previous list. The machine executed for the same 70 cycles and the timing data generated is shown below as 2530_8 or 1368 ns.

```
!000070 K  000000  000000  000000  000000  000000  000000  000000  000000
!REGISTERS,ADR=20000,TITLE FMT = . X
LOCAL STORE
     LS00    LS01    LS02    LS03    LS04    LS05    LS06    LS07
   000000  000000  000000  000000  000000  000000  000000  000000
     LS10    LS11    LS12    LS13    LS14    LS15    LS16    LS17
   000000  000000  000000  000000  000000  000000  000000  000000
     LS20    LS21    LS22    LS23    LS24    LS25    LS26    P.SYS
   000004  000001  000000  020000  000000  002550  020000  000000
    R.MX    R.IX    R.IY   R.MPC   R.ADR    FIDX    FMPC    FIST
   020001  000013  000000  002162  000000      20      33      00
!<LF>. X
```

The performance improved of both MERG1 machines is illustrated in Figure 7-22. A significant performance improved is obtained by tailoring the "MERG1" machine's architecture to a unique set of application criteria.



COMPARISON OF TIMINGS FOR MERGE MACHINES USING FACILITY

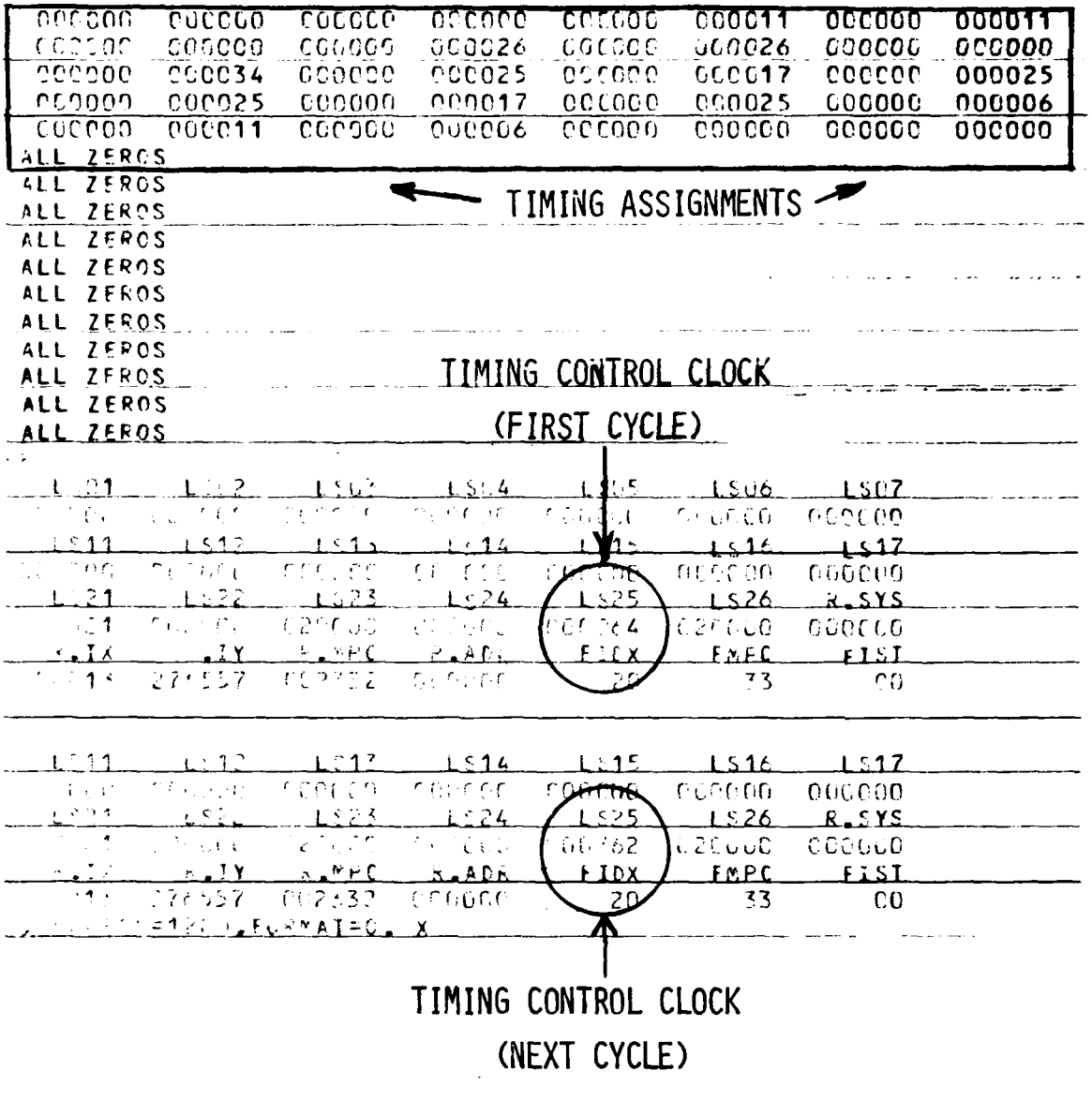
FIGURE 7-22

The modified "MERG1" machine requires 1368 ns and the original "MERG1" machine requires 2223 ns to produce the same output list for the identical input configuration. In both cases the tools are provided in the facility to conduct this type of analysis.

7.5.2.2 "MERG2" Machine

Next, the facility is used to generate data for a SPFA that performs the same DBMS function based on using a different merge algorithm than the "MERG1" machine. In this case, the "MERG2" machine was configured and executed. The "MERG2" machine was also described so that its fetch and mask processor, the compare processor and the coordination processor execute in parallel. A complete cycle of this machine produces two outputs for each merge compare operation. The input lists to be merged were identical to that configured for the "MERG1" machine. The actual output produced by the "MERG2" machine is identical to the output produced by the "MERG1" machine. Thus, the facility was used to execute two different merge machines where each performed the same DBMS function with the same results.

The first cycle of the "MERG2" machine produced output as illustrated in Figure 7-23 in 264_8 or 188 ns. However, at this point, all three processors are now executing in parallel. Thus, the time to produce the next output, also illustrated in Figure 7-23, is an accumulated total of 362_8 ns or a difference of 76_8 ns.



THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

TIMING RESULTS FOR MERG2 MACHINE FOR "OR" MERGE TYPE REQUEST

FIGURE 7-23

The "MERG2" machine was then executed for the same number of cycles needed to produce the identical output list that was produced by the "MERG1" machine for the same input lists as Figure 7-19. The timing data computed using the facility for the merge operation is 1922 ns. The performance of the "MERG2" machine as compared to the "MERG1" machine is also shown in Figure 7-22. The type of timing data collected illustrates how the different merge machines can be examined using the facility. These tests only illustrate the type of data that can be collected once the merge machines exist. A great deal of additional analysis can be conducted to study the composure of the input lists, and specific architectural constructs of each machine. For instance, the algorithm used for the "MERG2" machine compare processor can be expanded for additional parallelism. The difference in timing data collected, thus far, may indicate the degree of parallelism for the 2 X 2 merge network may not compensate for the added operations of its compare and coordinate processors. Therefore, added parallelism for the "MERG2" machines' compare processor can be considered by increasing the size of its merge network. The facility can be used to recreate the "MERG2" machine for larger networks and then be reconfigured for execution.

7.5.2.3 Additional Evaluations

The facility can also be used to examine details of the actual merging function using the SPFA machines. An example of this type of analysis is to utilize the machine to perform two different merge operation types using the same input lists. The "MERG1" machine was utilized for this analysis. The sample input list utilized are illustrated in Figure 7-24. The output of each merge type is also illustrated. For an AND, three output entries are produced and for an OR merge type, five output entries are produced. The AND of these lists requires 191ns while the OR requires 195 ns. The study of the actual merging operation indicates that the OR always produces an output entry while the AND may not.

This type of analysis is important to users wishing to study specific details of different merge algorithms for merge machines. Larger sets of inputs can be analyzed to study the effects of merge operation type on specific lists. The facility provides the capabilities to make these type of studies and to help analyze the merge operation for a specific application.

7.5.2.4 MERG3 Machine

The facility can also be utilized to tailor a SPFA to a specific application. For instance, by studying the execution flow of the "MERG1" and "MERG2" machines for special input lists, it can be shown that these machines keep recycling through the fetch and mask processor whenever input entries are not in correct context. The machines keep recycling until an acceptable entry is found from both lists for the "MERG1" machine and two acceptable entries are found from the input lists for the

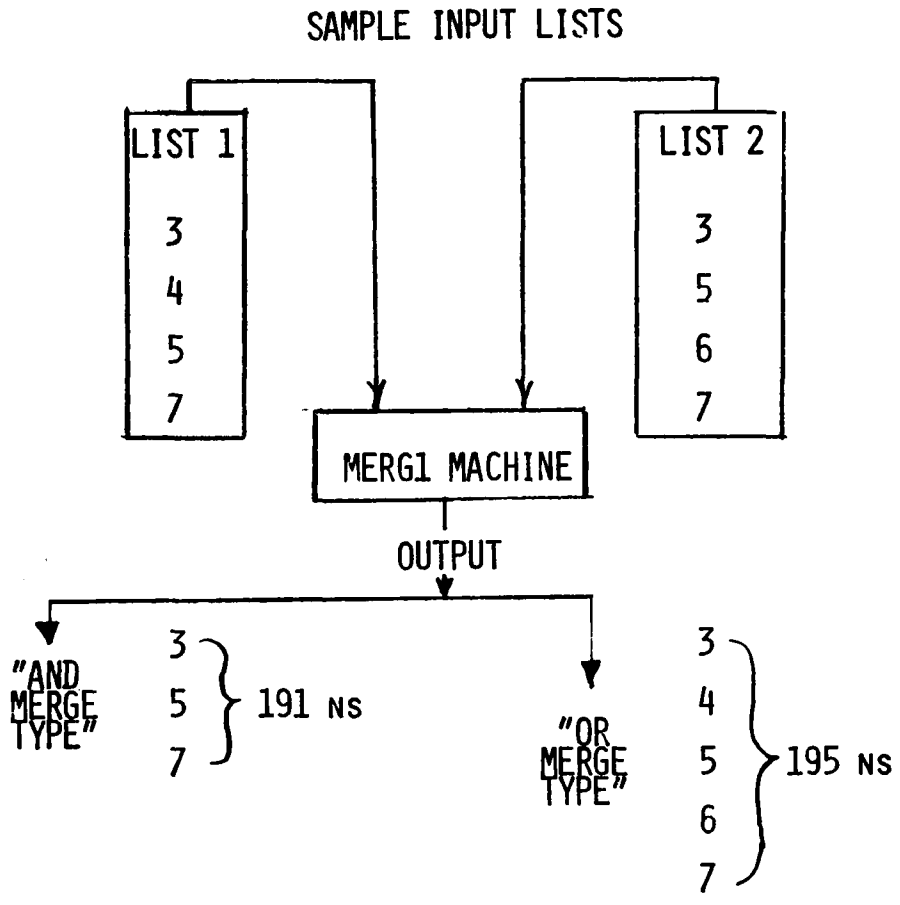
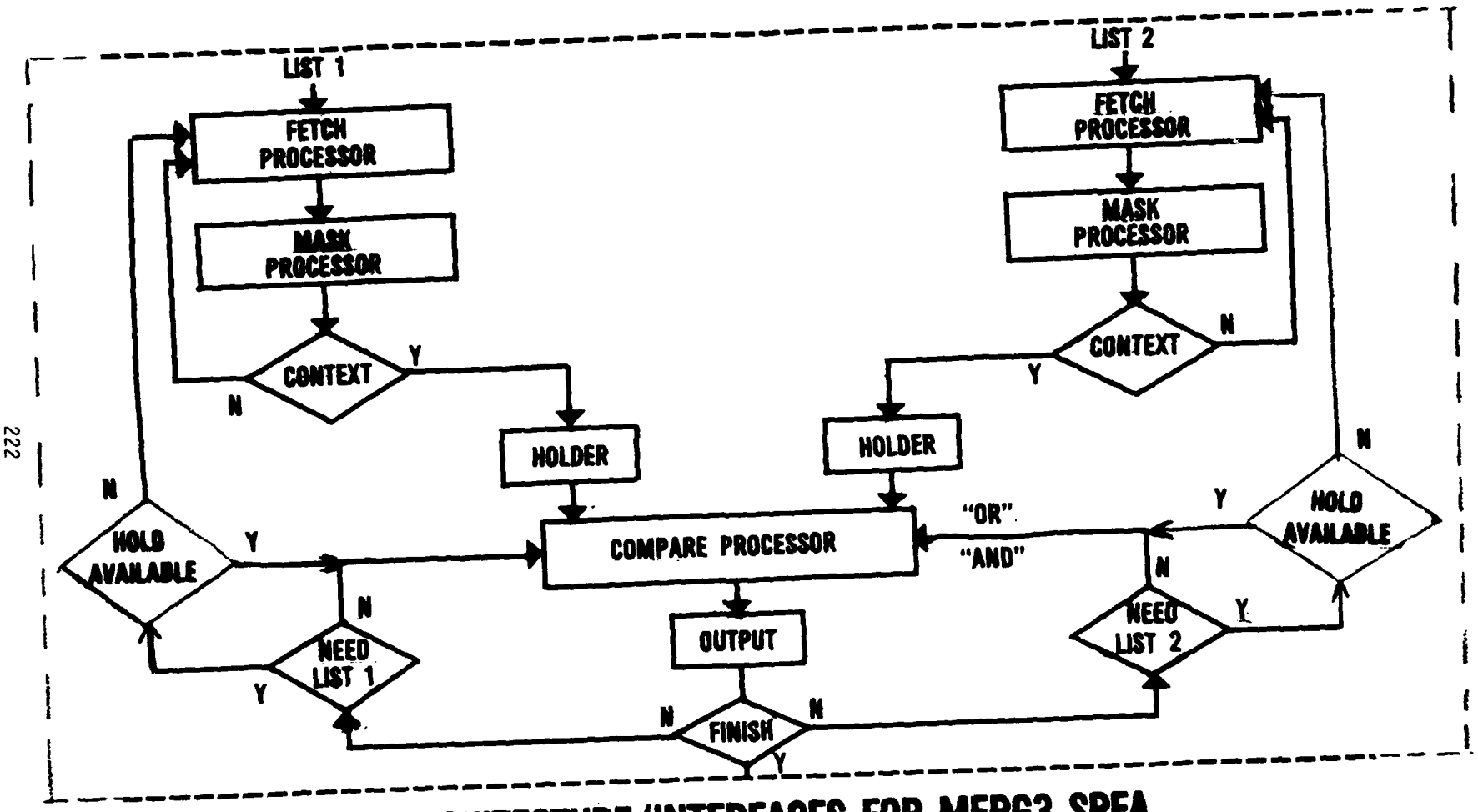


ILLUSTRATION OF USING FACILITY TO STUDY MERGE OPERATION TYPE

FIGURE 7-24

"MERG2" machine. Only when these entries are found and are presented to the compare processor does an actual merge occur. When the compare processor completes, an output entry may be formed and the fetch and mask cycle are repeated.

However, for special lists a different context checking algorithm may be devised. This algorithm requires the mask processor to examine the input context entries from each list and place acceptable entries into a bank of holding registers. The fetch and mask cycle is repeated until acceptable entries are found from both lists. However, during each fetch cycle entries are always fetched in parallel from both lists. Once acceptable entries are found from both lists, the compare processor merges the first entry in each bank of holding registers. Upon completion of this processor, the machine may not need to recycle to the fetch processor but may recycle to the compare processor if entries are still available in the holding registers. This may reduce the number of fetch and mask cycles for the machine for specific input lists. The facility was utilized to examine this algorithm by developing a "MERG3" SPFA with this modified mask checking algorithm. This machine is illustrated in Figure 7-25. The modification added a bank of holding registers to the "MERG1" SPFA description and added operations to the mask processor and modified the return of control following the compare processor so that the machine recycles to the compare processor to the fetch processor.



ARCHITECTURE/INTERFACES FOR MERG3 SPFA

FIGURE 7-25

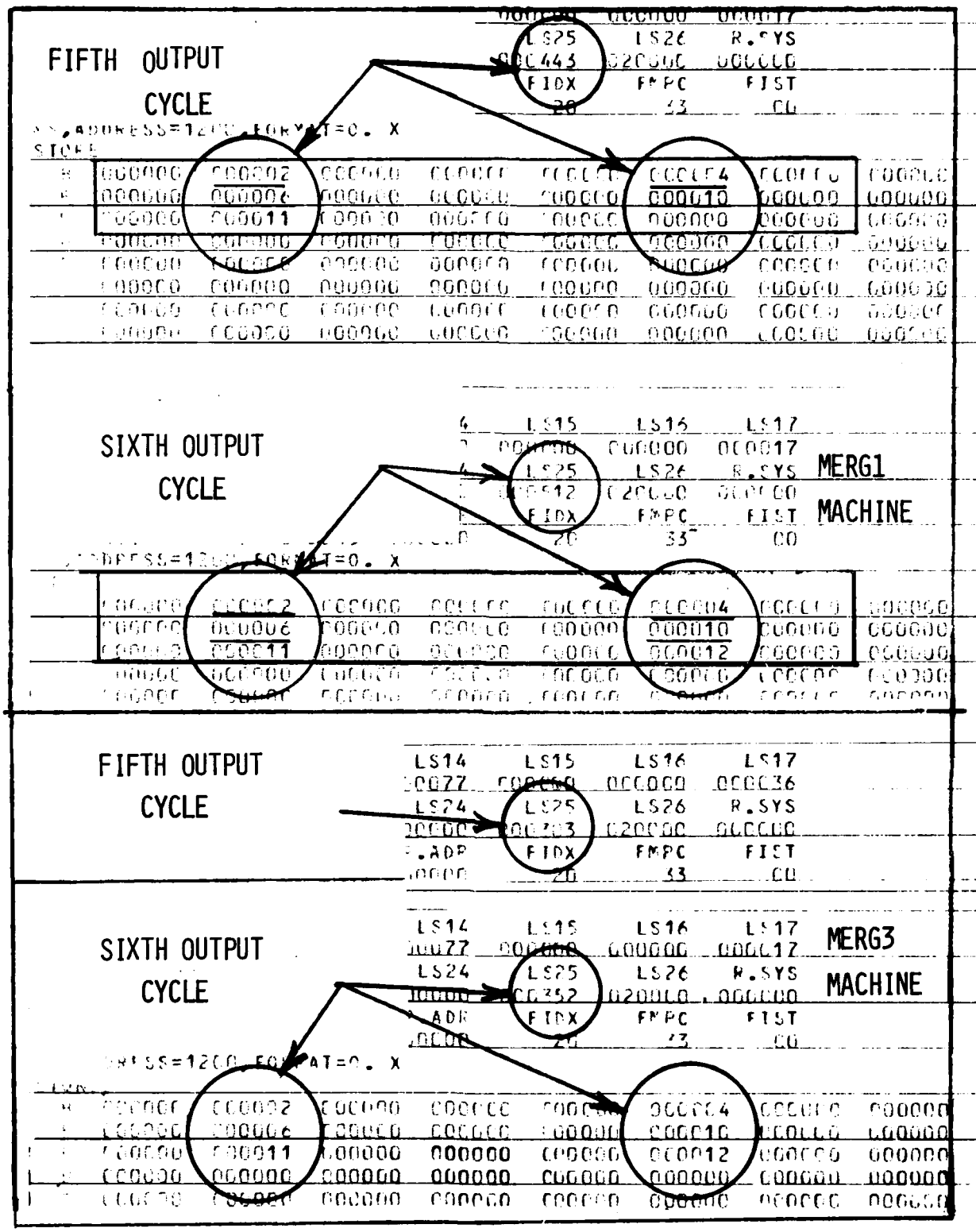
Both the "MERG1" and "MERG3" were executed using as input the specialized input lists illustrated in Figure 7-26. These inputs include:

LIST 1	LIST 2
20002	10001
20004	10003
20006	10005
20010	10007
20012	20011

The context of each entry is identified in column one. For this example a two in column one indicates a correct context. Only entries in correct context will be merged.

The first four entries of list two are not in correct context and will be rejected. The "MERG1" machine was executed and produced the output shown in Figure 7-26 as 2,4,6,10,11 for the fifth output cycle and 2,4,6,10,11,12 for the sixth cycle. For six cycles this machine required 512_8 or 330 ns. The "MERG3" machine was then configured with the same input lists. This machine was also executed for the same two cycles. As shown in Figure 7-26 the identical output is produced. However, this machine required, as illustrated in Figure 7-26, 352_8 or 234 ns for the sixth output cycle. Significant performance improvement was made by modifying the context checking algorithm of the "MERG1" machine. The facility user now must evaluate the changes to the machine required to gain the improved performance.

This test was conducted to illustrate how the facility can be utilized to modify a SPFA to accommodate a specific application criteria and then help gain performance timings based on this modification.



RESULTS OF COMPARING MERG1 AND MERG3 MACHINES FOR SAME MERGE OPERATIONS

FIGURE 7-26

In summary, the various performance tests conducted using the tools/components of the facility demonstrate typical types of capabilities available to facility users. This demonstration was also made to illustrate how the facility can be utilized to examine several types of tradeoffs when developing SPFA's.

First, the facility was utilized to examine the architectural features of different SPFAs based on their description and execution in the facility. Next, difference in these features were examined by assigning performance timing based on implementing the machine in hardware and executing the SPFA's as machines in the facility.

Once the SPFA machine was configured, the facility was used to examine how a range of characteristics that are part of a realization assumption are used to highlight sensitive portions of the machine. These, in turn, can be used to specify certain operations that may be implemented in using either high speed technology or highly reliable technology. Next, it is shown how these characteristics can also be used to generate timings for different machines.

Finally, it is shown how the facility can be used to reformat a description of a SPFA and redevelop it as a specialized SPFA machine. The type of tests conducted also illustrate how the facility can also be used as a vehicle to study specific details of other hardware algorithms that perform the merging operation. In addition,the facility can be used for other types of analysis that consist of:

(1) effects of varying input lists,

(2) variations in realization assumptions,

(3) creating additional specialized lists, and

(4) evaluating unique conditions for merging such as restart/recovery.

A final capability of the facility, that is demonstrated, is to use it to examine other approaches for performing the merge function. Some examples are now described.

7.6 OTHER COMPARISONS

Additional comparisons are also possible using the facility for other approaches that perform a candidate DBMS function. Two such approaches that have been examined are a merge routine that executes on an emulated M6800 microprocessor and uses direct MULTI micro-code to perform the merge function on the QM-1.

These approaches were examined because an emulation of the M6800 microprocessor existed on the QM-1 and the MULTI micro-code is available for the QM-1. Both approaches illustrate various types of considerations present when choosing alternative methods for current DBMS functions. The results of three approaches are illustrated in Table 7-1.

First, a performance range for each approach was established. The performance range is based on the time needed to complete a cycle of processing two input entries for a merge operation. This activity performed by a software merge routine that executes on an emulated M6800 microprocessor requires approximately 956 micro-seconds. The M6800 microprocessor was chosen because a M6800 emulation existed on the QM-1 and a meta compilier that produced code for the M6800 existed on the H6180/MULTICS system. The performance of the merge operation is based on the number of instruction cycles executed for the operation and a 1.9 cycle time for the M6800 microprocessor.

MERGE ROUTINE	EXECUTES ON	LANGUAGE USED	NUMBER INSTRUCTIONS REQUIRE	TIME TO PERFORM SPECIFIED MERGE OPERATIONS
M6800 MERGE ROUTINE	EMULATED M6800	M6800 MACHINE CODE	100	956 USEC.
QM-1 MERGE ROUTINE	QM-1	QM-1 MULTI	200	57.5 USEC.

PERFORMANCE CHARACTERISTICS OF MERGE FUNCTION USING VARIOUS MERGE METHODS

TABLE 7-1

A set of MULTI instructions were also developed to represent a micro-coded routine that can perform the same merge operation. This routine was developed to illustrate how the facility can be used to examine specialized considerations if an organization chooses to micro-code specific DBMS functions. The time to execute this routine on the QM-1 is approximately 57.6 micro-seconds. This time is based on the number of T-periods needed to execute the routine where each T-period on the QM-1 is 80 nano seconds.

An important consideration for illustrating these approaches is to illustrate how the facility is capable of providing, to potential users, the ability to generate performance data for competitive approaches an organization might consider for choosing candidate DBMS functions to be moved to hardware. The examination of these approaches can also provide data to help choose candidate DBMS functions to migrate from software to hardware. Several interesting factors can be examined because of the ability to develop these approaches in a common environment. For instance, programming complexity of new DBMS functions may be important to an organization lacking extensive development tools and programmers. The ability to examine various microprocessor machine coding conventions may be important to an organization with a large computing machine inventory. This type of data can be generated and examined by an organization to help decide if it has the resources necessary to develop either new software or micro-coded routine for an extensive DBMS upgrade program.

7.7 SUMMARY

The demonstration described in this chapter illustrates the types of capabilities available to users of a DMAD Facility. It is shown how the facility can be used to help choose DBMS functions to be moved into hardware and then actually demonstrated, using a specific facility, development of merge SPFA's. It is shown how the facility can be used to generate various types of timing data, modify description of a SPFA and examine data flow of executing merge SPFA machines. The use of the SMITE hardware description language represents a convenient way to document SPFA's, This documentation aide, used to describe SPFA's in a top-down structured manner, along with the timing data, illustrates a procedure that can provide inputs to help design hardware prototypes of SPFA's. Finally, the use of the QM-1 , as a microprogrammable emulation vehicle, is an important tool that can be used to execute SPFA's based on different architectural constructs, collect data and assist in the overall transition of DBMS functions from software into hardware.

CHAPTER VIII

DISCUSSION AND FUTURE RESEARCH

8.1 DISCUSSION

The research described in this report concerns the design of a data base machine development facility and the development of a methodology for the identification, creation, testing, evaluation and substitution of a special purpose function architecture. The development methodology utilizes the tools/components in the proposed DMAD Facility to help choose candidate DBMS functions to be moved from software into hardware and to evaluate one or more SPFA's that can perform the DBMS function. A specific instance of the facility is demonstrated using a QM-1 microprogrammable computer. This demonstration consists of using the proposed development methodology to develop several SPFA's that perform the same DBMS function which are based on different architectural constructs.

Described in CHAPTER IV is the overall facility specification and the development methodology. This specification includes the set of components that comprise the facility and a set of specialized SPFA development tools that are available to users of the facility. The components in the facility provide the:

(1) ability to execute a SPFA machine based on a functional description of its architecture,

(2) ability to support the execution of SPFA machines with a set of special resources,

(3) ability to identify the configuration requirements of machines executing in the facility, and

(4) capability to manage flow within the facility by assignment of its resources.

The specialized tools required in the facility would enable a user to perform a basic set of development processes. These processes would allow a user to tailor the development of a SPFA to a specific application. The methodology to utilize this facility requires an orderly selection of these processes during a SPFA's development. The processes are select candidate function, create, test, evaluate and substitute. For each process the facility is configured as a DBMS machine, a SPFA machine or a DBMS/SPFA machine. Once configured, the facility is essentially transformed into the desired machine. This transformed machine is available for user interaction, testing, execution and evaluation.

Addressed in CHAPTER V are the tools, components and procedures that are part of the methodology to develop SPFA's using the DMAD Facility. Included in this chapter is the introduction of concepts of a Process Description Language. This language is used to request a development process in the facility. Specific procedures in the facility are then followed to complete each process requested.

The select candidate function process utilizes the facility to collect utilization statistics on candidate DBMS software functions, provide a capability to dynamically test the execution paths of candidate functions, define a set of input and output interfaces for the function

and complete quality metrics for the function. This data is used to choose a candidate function which may be moved from software into hardware as a SPFA. The facility is then utilized to examine one or more SPFA's that can perform this function.

The origination of each SPFA is performed by the create process using the facility. This process results in a machine executable description of a SPFA. This description is used in the configuration of a SPFA machine. Once configured, the execution of a SPFA machine enables a user to invoke procedures for testing, evaluation and/or substitution.

The testing process insures the SPFA machine performs its intended DBMS function. This process is performed by verifying the execution path of a SPFA machine at various execution states. An evaluation process is performed by assigning timings for each SPFA's machine operations prior to its execution. These timings are assigned from a Realization Timing Control Array (RTCA). The timings are computed using realization assumptions that consist of a set of hardware characteristics that can be used to implement a SPFA. The performance of each SPFA machine is determined by accumulating these timings as each machine executes. These timings are used in the architecture evaluation of SPFA's and are used to help determine which SPFA best meets the requirements of an application.

Once the evaluation process is complete, the facility is used to assess the hardware/software tradeoffs of a SPFA's integration within a DBMS. The substitution process permits assessing the tradeoffs replacing a hardware DBMS function for a software DBMS function. This process permits assessing the overall impact and interface complexity

problems associated with the evolutionary introduction of a SPFA into a data base management system.

Identified in CHAPTER VI are a set of components that have been utilized to demonstrate the feasibility of a specific implementation of the DMAD Facility. In this implementation, a Nanodata QM-1 micro-programmable computer is used to execute several merge machines. These machines are based on designs of Hollaar's merge processor and on Stellhorn's inverted file processor.

Described in CHAPTER VII is a comprehensive set of procedures used to illustrate how the facility can be used to evaluate these MERGE SPFA machines. The results of this evaluation show how the facility is used:

(a) to examine specialized architectures that perform the *merge DBMS function*,

(b) establish a range of timing for the merge machine based on different hardware implementation strategies,

(c) examine details of specific merge algorithms for different merge request types, and

(d) tailor and modify a description of a SPFA to uniquely tailor its development to specific application criteria.

Finally, CHAPTER VII concludes with an examination of alternative approaches that can provide a merge DBMS function using other methods.

The major results of this report are summarized below;

(1) a methodology to develop DBMS special purpose function architectures is proposed as an organized and orderly approach consisting of a set of specialized processes,

(2) a set of components and tools designed as a Data Base Machine Architecture Development Facility can provide an environment that can be utilized to develop DBMS special purpose function architectures,

(3) specialized microprogrammable computers can be used as execution machines in a DMAD Facility and provide a specialized tool that can be utilized to examine hardware/software tradeoffs in the development of SPFA's,

(4) the measurement of the performance of DBMS special purpose function architectures (SPFA's) is feasible when they are described in a specialized high level hardware description language and executed on a microprogrammable computer, and

(5) various types of evaluation data can be collected using the set of tools/components of a specific DMAD Facility to demonstrate the feasibility of developing, examining and performing an architecture evaluation of several SPFA's that can perform a DBMS function.

8.2 FUTURE RESEARCH

The effect of continuing advancements in hardware technology is promoting the feasibility of having SPFA's perform many data base management functions. The DMAD Facility can serve as a specialized model to help determine which of these SPFA's should actually be hardware prototypes. The extensive use of such a facility requires that it have the capability to examine critical hardware/software tradeoff issues for choosing which DBMS functions should migrate from software to hardware. In order to utilize the facility in this role, several components/tools can be added or further extended to the facility to provide this capability.

These components/tools should be directed towards providing data to measure critical factors that enable a choice to be made for which DBMS functions should be moved from software to hardware. Further research is needed to fully define the components/tools needed. A process description language is proposed as a tool in the DMAD Facility to request SPFA development processes. This language can be expanded to help define these tools and special analysis tools capable of requesting tools in the facility. For instance, the language can be utilized to highlight a user's concern for pending hardware/software tradeoff's in his/her environment prior to choosing a SPFA. In this role, it may be necessary to be able to request analytical modeling techniques in the facility that may reduce the choice of SPFA's that actually need to be configured as machines in the facility.

The DMAD Facility, in this role, would be a source of several optimization methodologies that can be used to formulate a complete data base

machine. In order to assume this role, modeling techniques may be added as tools in the facility to refine the choice of SPFA machines. These techniques could examine competitive SPFA's and narrow the choices of SPFA machines that would be entered in the Data Base Machine Architecture Configuration Array (DMCA). Once several candidates are available as machines in the DMCA optimiation techniques are needed to judge the overall quality of several SPFA's for a specific application. In this mode, the DMCA would have to be automatically searched to specify a set of integrated SPFA machines that may comprise a specialized data base machine. Once a set of integrated SPFA machines are defined, the DMAD Facility can be configured as the candidate data base machine. In this mode, the facility needs a complete multiple machine emulation capability. Future research can be devoted to both identification of automated techniques to configure this machine for execution.

Another DMAD Facility tool that future research could enhance is the ability to identify and examine critical metrics that assist in highlighting hardware/software tradeoffs choosing SPFA's. The metrics calculated for the SPFA machine could be compared to the same metrics as the software module it replaces. An organization could then choose the SPFA that provides the highest metric rating for a specific application.

A methodology that incorporates several techniques to help choose candidate SPFA's including mathematical modeling, simulation and emulation using the DMAD Facility has been proposed by Vemuri (VEMU80). Further work is needed to automate specific features of this methodology including automating the use of the facility.

Finally, several additional topics for future research include:

(1) additional research for developing the actual logic circuitry for a SPFA's developed in the DMAD Facility from their high level hardware description language. This process should also provide procedures to include the factors for the SPFA prototype to meet specific application criteria using specific hardware implementation characteristics that are part of a realization assumption.

(2) automation of the assignments from the Realization Timing Control Array to insure performance data is generated for all possible realizations of the SPFA machine,

(3) additional studies to utilize advanced micro-programmable computers as execution machines in the DMAD Facility, with special emphasis on unique types of tools available on these machines to help tailor the SPFA being developed to specific criteria,

(4) a realized DMAD Facility may consist of an execution machine that provides some features of the QM-1. Studies are needed to examine further enhancements to the QM-1 as the forerunner of execution machines that can improve the efficiency, time and expense involved in using emulation as an approach to help develop special purpose function architectures.

(5) further research on the development of tools that can be utilized to study concurrency dependencies among the operations of a SPFA, and

(6) increased effort to study the effect of using the DMAD Facility to design a secure integrated distributed Data Base Machine.

APPENDIX A

PROCEDURE TO CREATE "MERG1" SPFA

The procedure to create the "MERG1" SPFA consists of describing the SPFA in the SMITE language. This description is documented in a MULTICS file called "MERG1,SMITE". This file is compiled using the SMITE compilier with the following command:

```
SMITE  - i  MERG1, SMITE
```

Following completion of the compiliation two files are produced, a source and object version. The source version is:

```
MERG1, LIST
```

and the object version is

```
MERG1 - L60
```

The source file is then examined for errors. If errors are found the source SPFA description is modified to correct the error. The source description is then re-compiled. When all errors have been eliminated the object version is down loaded to the QM-1. This is accomplished by initiating the NOVA emulator on the QM-1. A file is then created on the QM-1 to which the object file from MULTICS is transferred. This file is created by:

```
ADD CSPMERGE ADR = location, size: Type = Program, Execute
                                                   WRITE
```

Next, the following command is entered:

```
COPY   I =   TTC      O = /DOWN LOAD
```

This command initiates a signal to download the object file MERG1,L60 to the file DOWNLOAD on the QM-1.

Once the file is transferred it must be converted into a format needed to load the MERG1 SPFA description. The command:

LOADTAPE DOWNLOAD file 0 CSMERG1 EMULATOR

configure the machine in the format necessary in the file "CSMERG1". This file is configured on the QM-1 using the LOADCS command:

LOADCS CS MERG1

The following is the source listing of the "MERG1" SPFA created using this procedure.

MERG1

MACHINE

```
1     P-MERGE:  PROCESSOR;
2          DECLARE PROD-FLAGS<35:0> REGISTER,
2               STEP-FLAG FLAG DEFINED PROD-FLAGS<35>,
2               PC<15:0> DEFINED PROD-FLAGS<15:0>,
2                TIMER-VALUES[0:13]<15:0> REGISTER,
2                TIMER CLOCK,
2               REGS[0:15]<15:0> REGISTER,
2               STATUS DEFINED REGS[0],
2                   LISTSIZE-FLAG DEFINED STATUS<0>,
2                   ZOUTPUT-FLAG DEFINED STATUS<2>,
2                   NEED-XINPUT DEFINED STATUS<4>,
2                   NEED-YINPUT DEFINED STATUS<6>,
2                   INPUT-X DEFINED STATUS<8>,
2                   INPUT-Y DEFINED STATUS<10>,
2                      XINPUT-READY DEFINED STATUS<12>,
2                      YINPUT-READY DEFINED STATUS<14>,
2               X-REG DEFINED REGS[1],
2                Y-REG DEFINED REGS[2],
2                X-HOLD1 DEFINED REGS[3],
2                Y-HOLD1 DEFINED REGS[4],
2                COUNT-REG DEFINED REGS[5],
2                ZOUTPUT-REG1 DEFINED REGS[6],
2                   DOCNUMZ DEFINED ZOUTPUT-REG1<11:0>,
2                Z-REG DEFINED REGS[7],
2                LISTSIZE-XREG DEFINED REGS[8],
2                LISTSIZE-YREG DEFINED REGS[9],
2                I-HERE<4:0> REGISTER,
2                   X-INPUT1<15:0> REGISTER,
2                CONTXT1X<3:0> DEFINED X-INPUT1<15:12>,
2                DOCNUMX<11:0> DEFINED X-HOLD1<11:0>,
2                Y-INPUT1<15:0> REGISTER,
2                CONTXT1Y<3:0> DEFINED Y-INPUT1<15:12>,
2                DOCNUMY<11:0> DEFINED Y-HOLD1<11:0>,
2                IR<15:0> REGISTER,
2                OPCODE<2:0> DEFINED IR<2:0>,
2                CONTXT1<3:0> DEFINED IR<6:3>;
3                DECLARE
3                OP-STEP EXTERNAL,
3                OP-HALT EXTERNAL,
3                OP-ERROR EXTERNAL,
```

```
3                OP-NOTSIM EXTERNAL,
3                IN-PORT<15:0> PORT,
3                   OUT-PORT<15:0> PORT;
4            DECLARE
4                MEM[0:4095]<15:0> MEMORY;
5       INPUT-FETCH: PROCESSOR;
6           IN TIMER-VALUES[0] PARALLEL-BEGIN;
7               IF NEED-XINPUT THEN
7                    IN TIMER-VALUES[1] BEGIN;
8                   X-INPUT1 <- MEM[X-REG];
9                   X-REG <- X-REG +1;
10                      LISTSIZE-XREG <- LISTSIZE-XREG - 1;
11                     IF LISTSIZE-XREG = 0 THEN
11                      LISTSIZE-FLAG <- 1;
12                      END IF;
13                     NEED-XINPUT <- 0;
14                     XINPUT-READY <- 1;
15                  END;
16              END IF;
17             IF NEED-YINPUT THEN
17                   IN TIMER-VALUES[2] BEGIN;
18                    Y-INPUT1 <- MEM[Y-REG];
19                     Y-REG <- Y-REG +1;
20                     LISTSIZE-YREG <- LISTSIZE-YREG - 1;
21                     IF LISTSIZE-YREG = 0 THEN
21                     LISTSIZE-FLAG <- 1;
22                    END IF;
23                     NEED-YINPUT <- 0;
24                     YINPUT-READY <- 1;
25                  END;
26              END IF;
27          PARALLEL-END;
28      INPUT-FETCH: END;
29      MASK: PROCESSOR;
30           IN TIMER-VALUES[3] PARALLEL-BEGIN;
31               IF XINPUT-READY THEN
31                   IN TIMER-VALUES[4] BEGIN;
32                      IF (CONTXT1X = CONTXT1)
32                         THEN BEGIN;
33                           X-HOLD1 <- X-INPUT1;
```

```
34                      XINPUT-READY <- 0;
35                      INPUT-X <- 1;
36                      END;
37                      ELSE BEGIN;
38                      XINPUT-READY <- 0;
39                      NEED-XINPUT <- 1;
40                          END;
41                       END IF;
42                END;
43                END IF;
44               IF YINPUT-READY THEN
44                    IN TIMER-VALUES[5]  BEGIN;
45                       IF (CONTXT1Y = CONTXT1)
45                          THEN BEGIN;
46                             Y-HOLD1 <- Y-INPUT1;
47                          INPUT-Y <- 1;
48                             YINPUT-READY <- 0;
49                          END;
50                          ELSE BEGIN;
51                             YINPUT-READY <- 0;
52                          NEED-YINPUT <- 1;
53                          END;
54                       END IF;
55                END;
56           END IF;
57           PARALLEL-END;
58      MASK:  END;
59      COMPARE:  PROCESSOR;
60            CASE OPCODE;
61                IN TIMER-VALUES[6] PARALLEL-BEGIN;
62                   IF DOCNUMX < DOCNUMY THEN
62               IN TIMER-VALUES[7] BEGIN;
63                          NEED-XINPUT <- 1;
64                          INPUT-X <- 0;
65                      END;
66                   END IF;
67                   IF DOCNUMX = DOCNUMY  THEN
67                   IN TIMER-VALUES[8] BEGIN;
68                         DOCNUMZ <- DOCNUMX;
69                          OUTPUT-Z;
```

```
 70                                  NEED-XINPUT <- 1;
 71                          NEED-YINPUT <-1;
 72                          INPUT-X <- 0;
 73                          INPUT-Y <- 0;
 74                          ZOUTPUT-FLAG <- 1;
 75                        END;
 76                      END IF;
 77                    IF DOCNUMX > DOCNUMY THEN
 77                     IN TIMER-VALUES[9] BEGIN;
 78                          NEED-YINPUT <- 1;
 79                        INPUT-Y <- 0;
 80                        END;
 81                    END IF;
 82                PARALLEL-END;
 83                 IN TIMER-VALUES[10] PARALLEL-BEGIN;
 84                    IF DOCNUMX < DOCNUMY THEN
 84                      IN TIMER-VALUES[11] BEGIN;
 85                          DOCNUMZ <- DOCNUMX;
 86                          OUTPUT-Z;
 87                      NEED-XINPUT <- 1;
 88                         INPUT-X <- 0;
 89                          ZOUTPUT-FLAG <- 1;
 90                        END;
 91                    END IF;
 92                    IF DOCNUMX = DOCNUMY THEN
 92                       IN TIMER-VALUES[12] BEGIN;
 93                          DOCNUMZ <- DOCNUMX;
 94                    OUTPUT-Z;
 95                          NEED-XINPUT <- 1;
 96                        NEED-YINPUT <- 1;
 97                          INPUT-X <- 0;
 98                          INPUT-Y <- 0;
 99                          ZOUTPUT-FLAG <- 1;
100                       END;
101                    END IF;
102                    IF DOCNUMX > DOCNUMY THEN
102                  IN TIMER-VALUES[13] BEGIN;
103                          DOCNUMZ <- DOCNUMY;
104                          OUTPUT-Z;
105                        NEED-YINPUT <- 1;
```

```
106                         INPUT-Y <- 0;
107                         ZOUTPUT-FLAG <- 1;
108                       END;
109                    END IF;
110                PARALLEL-END;
111                 OP-NOTSIM;
112                 OP-NOTSIM;
113                 OP-NOTSIM;
114                 OP-NOTSIM;
115                 OP-NOTSIM;
116                 OP-NOTSIM;
117              END CASE;
118      COMPARE:   END;
119           OUTPUT-Z:   PROCESSOR;
120                    BEGIN;
121                MEM[Z-REG] <- ZOUTPUT-REG1;
122                Z-REG <- Z-REG + 1;
123                END;
124           OUTPUT-Z: END;
125            PC <- 0;
126          DO FOREVER; BEGIN;
128                IF STEP-FLAG
128                    THEN BEGIN;
129                        OP-STEP;
130                    END;
131                END IF;
132                IR <- MEM[PC];
133                 X-REG <- MEM[PC+1];
134                 Y-REG <- MEM[PC+2];
135                 LISTSIZE-XREG <- MEM[PC+3];
136                 LISTSIZE-YREG <- MEM[PC+4];
137                 Z-REG <- MEM[PC+5];
138                 PC <- PC+6 ;
139                 NEED-XINPUT <- 1;
140                 NEED-YINPUT <- 1;
141                 DO UNTIL LISTSIZE-FLAG;
142                    BEGIN;
143                      ZOUTPUT-FLAG <- 0;
144                        IF STEP-FLAG
144                        THEN BEGIN;
```

```
145                         OP-STEP;
146                       END;
147                     END IF;
148                        DO UNTIL ZOUTPUT-FLAG;
149                           BEGIN;
150                           DO UNTIL INPUT-X AND INPUT-Y;
151                              BEGIN;
152                                 INPUT-FETCH;
153                                 MASK;
154                               END;
155                               COMPARE;
156                           END;
157                            END;
158                       END;
159           P-MERGE:    END;
```

APPENDIX B

This appendix contains the source listing of the "MERG2" SPFA created using the same procedures described in Appendix A.

```
1     OE-MERGE:  PROCESSOR;                                    MERG2
2          DECLARE PROD-FLAGS<35:0> REGISTER,
2            STEP-FLAG FLAG DEFINED PROD-FLAGS<35>,           MACHINE
2            PC <15:0> DEFINED PROD-FLAGS<15:0>,
2            TIMER-VALUE[0:19]<15:0> REGISTER,
2            TIMER CLOCK,
2             X-REG<15:0> REGISTER,
2             Y-REG<15:0> REGISTER,
2             Z-REG<15:0> REGISTER,
2            REGS[0:9]<15:0> REGISTER,
2          STATUS DEFINED REGS[0],
2            LISTSIZE-FLAG DEFINED STATUS<0>,
2            ZOUTPUT-FLAG DEFINED STATUS<2>,
2            NEED-XINPUT DEFINED STATUS<4>,
2            NEED-YINPUT DEFINED STATUS<6>,
2            XINPUT-READY DEFINED STATUS<8>,
2            YINPUT-READY DEFINED STATUS<10>,
2            INPUT-X DEFINED STATUS<12>,
2            INPUT-Y DEFINED STATUS<14>,
2            FIN DEFINED STATUS<15>,
2          L1 DEFINED REGS[1],
2          L2 DEFINED REGS[2],
2          LISTSIZE-XREG DEFINED REGS[3],
2          LISTSIZE-YREG DEFINED REGS[4],
2          COOR-HOLD DEFINED REGS[5],
2          X-INPUT DEFINED REGS[6],
2          Y-INPUT DEFINED REGS[7],
2          CONT DEFINED REGS[8],
2          CONTXTX<3:0> DEFINED X-INPUT<15:12>,
2          DOCNUMX<11:0> DEFINED X-INPUT<11:0>,
2          CONTXTY<3:0> DEFINED Y-INPUT<15:12>,
2          DOCNUMY<11:0> DEFINED Y-INPUT<11:0>,
2           REG-PAR DEFINED REGS[9],
2           NEED-INPUT DEFINED REG-PAR<0>,
2           NEED-COMPARE DEFINED REG-PAR<2>,
2           NEED-COORDINATE DEFINED REG-PAR<4>;
3     DECLARE
3
3          LOWL1<15:0> REGISTER,
3          HIGHL1<15:0> REGISTER,
```

```
3           LOW12<15:0> REGISTER,
3           HIGHL2<15:0> REGISTER,
3           OO-HOLD1<15:0> REGISTER,
3           EE-HOLD1<15:0> REGISTER,
3           HOLDL1[0:3]<15:0> REGISTER,
3           HOLDL2[0:3]<15:0> REGISTER,
3           O-HOLD1 DEFINED HOLDL1[0],
3           O-HOLD2 DEFINED HOLDL2[0],
3           O-HOLD3 DEFINED HOLDL1[2],
3           O-HOLD4 DEFINED HOLDL2[2],
3           E-HOLD1 DEFINED HOLDL1[1],
3           E-HOLD2 DEFINED HOLDL2[1],
3           IR<15:0> REGISTER,
3           OPCODE<2:0> DEFINED IR<2:0>,
3           CONTXT1<3:0> DEFINED IR<6:3>;
4       DECLARE
4           OP-STEP EXTERNAL,
4           OP-HALT EXTERNAL,
4           OP-ERROR EXTERNAL,
4           OP-NOTSIM EXTERNAL,
4           IN-PORT<15:0> PORT,
4           OUT-PORT<15:0> PORT;
5       DECLARE
5           MEM[0:4095]<15:0> MEMORY;
6       INPUT-FETCH: PROCESSOR;
7           IN TIMER-VALUE[0] PARALLEL-BEGIN;
8             IF NEED-XINPUT THEN
8             IN TIMER-VALUE[1] BEGIN;
9             X-INPUT <-MEM[X-REG];
10              X-REG <-X-REG+1;
11            LISTSIZE-XREG <-LISTSIZE-XREG - 1;
12              IF LISTSIZE-XREG=0 THEN
12                LISTSIZE-FLAG <-1;
13              END IF;
14            NEED-XINPUT <-0;
15            XINPUT-READY <-1;
16            END;
17          END IF;
18          IF NEED-YINPUT THEN
18            IN TIMER-VALUE[2] BEGIN;
```

```
19              Y-INPUT <- MEM[Y-REG];
20              Y-REG <-Y-REG+1;
21              LISTSIZE-YREG <-LISTSIZE-YREG -1;
22            IF LISTSIZE-YREG=0 THEN
22                LISTSIZE-FLAG <-1;
23            END IF;
24              NEED-YINPUT <-0;
25              YINPUT-READY <-1;
26                    END;
27                 END IF;
28              PARALLEL-END;
29            INPUT-FETCH: END;
30       MASK: PROCESSOR;
31            IN TIMER-VALUE[3] PARALLEL-BEGIN;
32              IF XINPUT-READY THEN
32              IN TIMER-VALUE[4] BEGIN;
33                 IF CONTXTX=CONTXT1
33                   THEN BEGIN;
34                     HOLDL1[L1] <- DOCNUMX;
35                       L1 <- L1+1;
36                     IF L1=2
36                       THEN BEGIN;
37                       XINPUT-READY <-0;
38                       INPUT-X <-1;
39                       L1 <-0;
40                       END;
41                     ELSE BEGIN;
42                       NEED-XINPUT <-1;
43                       XINPUT-READY <-0;
44                       END;
45                     END IF;
46                   END;
47                 ELSE BEGIN;
48                   XINPUT-READY <-0;
49                   NEED-XINPUT <-1;
50                   END;
51               END IF;
52             END;
53            END IF;
54               IF YINPUT-READY THEN
```

```
            IN TIMER-VALUE[5] BEGIN;
               IF CONTXTY = CONTXT1
                 THEN BEGIN;
                   HOLDL2[L2] <- DOCNUMY;
                     L2 <- L2+1;
                   IF L2=2
                     THEN BEGIN;
                       YINPUT-READY <-0;
                       INPUT-Y <-1;
                       L2 <- 0;
                       END;
                     ELSE BEGIN;
                       NEED-YINPUT <-1;
                       YINPUT-READY <-0;
                       END;
                     END IF;
                  END;
                 ELSE BEGIN;
                   YINPUT-READY <-0;
                   NEED-YINPUT <-1;
                   END;
                 END IF;
              END;
     END IF;
     PARALLEL-END;
     MASK: END;
     COMPARE: PROCESSOR;
     IN TIMER-VALUE[6] PARALLEL-BEGIN;
          COMP (O-HOLD1, O-HOLD2);
          COMP (E-HOLD1, E-HOLD2);
          PARALLEL-END;
          COMP (E-HOLD1, O-HOLD2);
        COMPARE: END;
     COMP: PROCESSOR(I1,I2);
          DECLARE I1<15:0> REGISTER,I2<15:0> REGISTER,
               TEMP<15:0> REGISTER;
     IN TIMER-VALUE[7] IF I2 < I1 THEN
            BEGIN;
               TEMP <- I2;
               I2 <- I1;
```

```
 90                I1 <- TEMP;
 91              END;
 92            END IF;
 93          COMP: END;
 94        COORDINATE: PROCESSOR;
 95            CASE OPCODE;
 96        IN TIMER-VALUE[8] BEGIN;
 97            IF COOR-HOLD = OO-HOLD1 THEN
 97            IN TIMER-VALUE[9] BEGIN;
 98                MEM[Z-REG] <- OO-HOLD1;
 99                Z-REG <- Z-REG+1;
100               COOR-HOLD <- EE-HOLD1;
101               END;
102            ELSE IN TIMER-VALUE[10] BEGIN;
103              IF OO-HOLD1 = EE-HOLD1
103              THEN BEGIN;
104              MEM[Z-REG] <- OO-HOLD1;
105                  Z-REG <- Z-REG+1;
106                END;
107              ELSE BEGIN;
108                COOR-HOLD <- EE-HOLD1;
109                END;
110              END IF;
111            END;
112         END IF;
113        END;
114        IN TIMER-VALUE[11] BEGIN;
115            IF COOR-HOLD = OO-HOLD1 THEN
115            IN TIMER-VALUE[12] BEGIN;
116                MEM[Z-REG] <- OO-HOLD1;
117                  Z-REG <- Z-REG+1;
118                  COOR-HOLD <- EE-HOLD1;
119                END;
120            ELSE IN TIMER-VALUE[13] BEGIN;
121                MEM[Z-REG] <- COOR-HOLD;
122                  Z-REG <- Z-REG+1;
123                  IF OO-HOLD1 = EE-HOLD1
123                    THEN BEGIN;
124                    MEM[Z-REG] <- OO-HOLD1;
125                    COOR-HOLD <- EE-HOLD1;
```

```
126                    Z-REG <- Z-REG +1;
127                    END;
128                  ELSE BEGIN;
129                   MEM[Z-REG] <- OO-HOLD1;
130                   Z-REG <- Z-REG+1;
131                   COOR-HOLD <- EE-HOLD1;
132                  END;
133                END IF;
134              END;
135           END IF;
136           END;
137           OP-NOTSIM;
138            OP-NOTSIM;
139           OP-NOTSIM;
140           OP-NOTSIM;
141           OP-NOTSIM;
142           OP-NOTSIM;
143           END CASE;
144      COORDINATE: END;
145       OUTPUT-L1: PROCESSOR;
146            IN TIMER-VALUE[17] BEGIN;
147            MEM[Z-REG] <- HOLDL1[1];
148              Z-REG <- Z-REG+1;
149           MEM[Z-REG] <- HOLDL1[2];
150              Z-REG <- Z-REG+1;
151           DO UNTIL LISTSIZE-FLAG;
152              BEGIN;
153           X-INPUT <- MEM[X-REG];
154              X-REG <- X-REG+1;
155           LISTSIZE-XREG <- LISTSIZE-XREG -1;
156              IF LISTSIZE-XREG=0 THEN
156                LISTSIZE-FLAG <-1;
157              END IF;
158              IF CONTXTX = CONTXT1
158              THEN BEGIN;
159                MEM[Z-REG] <- X-INPUT;
160                  Z-REG <- Z-REG+1;
161                END;
162              END IF;
163           END;
```

```
164           END;
165         OUTPUT-L1: END;
166       OUTPUT-L2: PROCESSOR;
167             IN TIMER-VALUE[18] BEGIN;
168             MEM[Z-REG]  <- HOLDL2[1];
169               Z-REG <- Z-REG+1;
170             MEM[Z-REG] <- HOLDL2[2];
171               Z-REG <- Z-REG+1;
172               DO UNTIL LISTSIZE-FLAG;
173                 BEGIN;
174                 Y-INPUT <- MEM[Y-REG];
175                   Y-REG <- Y-REG+1;
176               LISTSIZE-YREG <- LISTSIZE-YREG -1;
177                   IF LISTSIZE-YREG=0 THEN
177                     LISTSIZE-FLAG <- 1;
178                   END IF;
179               IF CONTXTY = CONTXT1
179                 THEN BEGIN;
180                 MEM[Z-REG] <- Y-INPUT;
181                 Z-REG <- Z-REG+1;
182                 END;
183               END IF;
184             END;
185             END;
186       OUTPUT-L2: END;
187     PC <- 0;
188     FIN <- 0;
189     DO UNTIL FIN;
190           BEGIN;
191           IF STEP-FLAG
191           THEN BEGIN;
192             OP-STEP;
193           END;
194         END IF;
195         IR <- MEM[PC];
196         X-REG <- MEM[PC+1];
197         Y-REG <- MEM[PC+2];
198         LISTSIZE-XREG <- MEM[PC+3];
199         LISTSIZE-YREG <- MEM[PC+4];
200         Z-REG <- MEM[PC+5];
```

```
231          PC <- PC+6;
232          NEED-XINPUT <- 1;
233          NEED-YINPUT <- 1;
234          NEED-INPUT <- 1;
235       DO UNTIL LISTSIZE-FLAG;
236            BEGIN;
207            IF LISTSIZE-XREG /=0 THEN BEGIN;
208              ZOUTPUT-FLAG <- 0;
209             IF STEP-FLAG
209             THEN BEGIN;
210             OP-STEP;
211             END;
212             END IF;
213             END;
214           ELSE BEGIN;
215             IF LISTSIZE-YREG /=0
215               THEN BEGIN;
216                 LISTSIZE-FLAG <- 0;
217                 OUTPUT-L2;
218                 FIN <- 1;
219                 END;
220       ELSE FIN <- 1;
221               END IF;
222             END;
223           END IF;
224           IF LISTSIZE-YREG /=0 THEN BEGIN;
225             ZOUTPUT-FLAG <- 0;
226            IF STEP-FLAG
226            THEN BEGIN;
227            OP-STEP;
228            END;
229            END IF;
230            END;
231           ELSE BEGIN;
232             IF LISTSIZE-XREG /= 0
232               THEN BEGIN;
233                 LISTSIZE-FLAG <-0;
234                 OUTPUT-L1;
235                 FIN <-1;
236                 END;
```

```
237      ELSE FIN <- 1;
238              END IF;
239            END;
240          END IF;
241      DO UNTIL ZOUTPUT-FLAG;
242            PARALLEL-BEGIN;
243            IF NEED-INPUT
243            THEN BEGIN;
244          DO UNTIL INPUT-X AND INPUT-Y;
245            BEGIN;
246              INPUT-FETCH;
247              MASK;
248            END;
249          IN TIMER-VALUE[14] BEGIN;
250           NEED-COMPARE <- 1;
251          O-HOLD1 <- HOLDL1[0];
252
252          E-HOLD1 <- HOLDL1[1];
253          O-HOLD2 <- HOLDl2[0];
254          E-HOLD2 <- HOLDL2[1];
255          LOWL1 <- O-HOLD1;
256
256          HIGHL1 <- E-HOLD1;
257          LOWL2 <- O-HOLD2;
258          HIGHL2 <- E-HOLD2;
259          END;
260           END;
261           END IF;
262           IF NEED-COMPARE
262           THEN BEGIN;
263            COMPARE;
264           IN TIMER-VALUE[15] BEGIN;
265           NEED-COORDINATE <- 1;
266          OO-HOLD1 <- O-HOLD1;
267          EE-HOLD1 <- E-HOLD1;
268          IF HIGHL1 = E-HOLD2
268          THEN BEGIN;
269             IF LOWL1 = O-HOLD2
269               THEN BEGIN;
270               O-HOLD1 <- O-HOLD2;
```

```
271                O-HOLD2 <- E-HOLD2;
272
272                INPUT-Y <- 0;
273                NEED-YINPUT <- 1;
274                YINPUT-READY <- 1;
275                NEED-INPUT <- 1;
276                END;
277                ELSE BEGIN;
278                E-HOLD1 <- O-HOLD2;
279                L1 <- L1 +1;
280                O-HOLD1 <- E-HOLD2;
281                L2 <- L2 + 1;
282                INPUT-X <- 0;
283                INPUT-Y <- 0;
284                NEED-XINPUT <- 1;
285                NEED-YINPUT <- 1;
286                XINPUT-READY <- 0;
287                YINPUT-READY <- 0;
288                NEED-INPUT <- 1;
289                END;
290             END IF;
291          END;
292          END IF;
293          IF HIGHL2 = E-HOLD2
293          THEN BEGIN;
294               IF LOWL2 = O-HOLD2
294               THEN BEGIN;
295                E-HOLD1 <- O-HOLD2;
296               INPUT-X <- 0;
297               NEED-XINPUT <- 1;
298               XINPUT-READY <- 0;
299               NEED-INPUT <- 1;
300               END;
301               ELSE BEGIN;
302               O-HOLD1 <- O-HOLD2;
303               L1 <- L1 + 1;
304               O-HOLD2 <- E-HOLD2;
305               L2 <- L2 + 1;
306               INPUT-X <- 0;
307               INPUT-Y <- 0;
```

```
308                NEED-XINPUT <- 1;
309                NEED-YINPUT <- 1;
310                XINPUT-READY <- 0;
311                 YINPUT-READY <- 0;
312                NEED-INPUT <- 1;
313                END;
314                END IF;
315             END;
316             END IF;
317             END;
318              END;
319              END IF;
320              IF NEED-COORDINATE
320              THEN BEGIN;
321             COORDINATE;
322             IN TIMER-VALUE[16] BEGIN;
323             ZOUTPUT-FLAG <- 1;
324             NEED-INPUT <- 1;
325             END;
326             END;
327                   END IF;
328             PARALLEL-END;
329
329               END;
330             END;
331      OE-MERGE:END;
```

APPENDIX C

This appendix contains a listing of the modified MERG1 machine. The description is modified between statements 152 and 166 so the Fetch and Mask processor execute in parallel with the compare processor.

```
1     P-MERGE:  PROCESSOR;
2          DECLARE PROD-FLAGS<35:0> REGISTER,
2               STEP-FLAG FLAG DEFINED PROD-FLAGS<35>,
2               PC<15:0> DEFINED PROD-FLAGS<15:0>,
2                TIMER-VALUES[0:13]<15:0> REGISTER,
2                TIMER CLOCK,
2               REGS[0:15]<15:0> REGISTER,
2               STATUS DEFINED REGS[0],
2                   LISTSIZE-FLAG DEFINED STATUS<0>,
2                   ZOUTPUT-FLAG DEFINED STATUS<2>,
2                   NEED-XINPUT DEFINED STATUS<4>,
2                   NEED-YINPUT DEFINED STATUS<6>,
2                   INPUT-X DEFINED STATUS<8>,
2                   INPUT-Y DEFINED STATUS<10>,
2                   NEED-COMPARE DEFINED STATUS<11>,
2
2                      XINPUT-READY DEFINED STATUS<12>,
2                      YINPUT-READY DEFINED STATUS<14>,
2                  NEED-INPUT DEFINED STATUS<15>,
2               X-REG DEFINED REGS[1],
2                Y-REG DEFINED REGS[2],
2                X-HOLD1 DEFINED REGS[3],
2                Y-HOLD1 DEFINED REGS[4],
2                COUNT-REG DEFINED REGS[5],
2                ZOUTPUT-REG1 DEFINED REGS[6],
2                   DOCNUMZ DEFINED ZOUTPUT-REG1<11:0>,
2                Z-REG DEFINED REGS[7],
2                LISTSIZE-XREG DEFINED REGS[8],
2                LISTSIZE-YREG DEFINED REGS[9],
2                X-IN DEFINED REGS[10],
2                Y-IN DEFINED REGS[11],
2                I-HERE<4:0> REGISTER,
2                   X-INPUT1<15:0> REGISTER,
2                CONTXT1X<3:0> DEFINED X-INPUT1<15:12>,
2                DOCNUMX<11:0> DEFINED X-HOLD1<11:0>,
2                Y-INPUT1<15:0> REGISTER,
2                CONTXT1Y<3:0> DEFINED Y-INPUT1<15:12>,
2                DOCNUMY<11:0> DEFINED Y-HOLD1<11:0>,
2                IR<15:0> REGISTER,
2                OPCODE<2:0> DEFINED IR<2:0>,
```

MODIFIED
MERG1
MACHINE

```
2                CONTXT1<3:0> DEFINED IR<6:3>;
3                DECLARE
3                OP-STEP EXTERNAL,
3                OP-HALT EXTERNAL,
3                OP-ERROR EXTERNAL,
3                OP-NOTSIM EXTERNAL,
3                IN-PORT<15:0> PORT,
3                   OUT-PORT<15:0> PORT;
4             DECLARE
4                MEM[0:4095]<15:0> MEMORY;
5        INPUT-FETCH: PROCESSOR;
6            IN TIMER-VALUES[0] PARALLEL-BEGIN;
7                IF NEED-XINPUT THEN
7                     IN TIMER-VALUES[1] BEGIN;
8                    X-INPUT1 <- MEM[X-REG];
9                    X-REG <- X-REG +1;
10                       LISTSIZE-XREG <- LISTSIZE-XREG - 1;
11                      IF LISTSIZE-XREG = 0 THEN
11                       LISTSIZE-FLAG <- 1;
12                       END IF;
13                      NEED-XINPUT <- 0;
14                      XINPUT-READY <- 1;
15                   END;
16               END IF;
17              IF NEED-YINPUT THEN
17                    IN TIMER-VALUES[2] BEGIN;
18                    Y-INPUT1 <- MEM[Y-REG];
19                     Y-REG <- Y-REG +1;
20                     LISTSIZE-YREG <- LISTSIZE-YREG - 1;
21                     IF LISTSIZE-YREG = 0 THEN
21                     LISTSIZE-FLAG <- 1;
22                    END IF;
23                     NEED-YINPUT <- 0;
24                     YINPUT-READY <- 1;
25                   END;
26               END IF;
27            PARALLEL-END;
28       INPUT-FETCH: END;
29       MASK: PROCESSOR;
30            IN TIMER-VALUES[3] PARALLEL-BEGIN;
```

```
31                IF XINPUT-READY THEN
31                    IN TIMER-VALUES[4] BEGIN;
32                       IF (CONTXT1X = CONTXT1)
32                          THEN BEGIN;
33                            X-IN <- X-INPUT1;
34                       XINPUT-READY <- 0;
35                       INPUT-X <- 1;
36                       END;
37                       ELSE BEGIN;
38                       XINPUT-READY <- 0;
39                       NEED-XINPUT <- 1;
40                          END;
41                        END IF;
42                END;
43                END IF;
44               IF YINPUT-READY THEN
44                    IN TIMER-VALUES[5]  BEGIN;
45                       IF (CONTXT1Y = CONTXT1)
45                          THEN BEGIN;
46                             Y-IN <- Y-INPUT1;
47                          INPUT-Y <- 1;
48                             YINPUT-READY <- 0;
49                          END;
50                          ELSE BEGIN;
51                             YINPUT-READY <- 0;
52                          NEED-YINPUT <- 1;
53                          END;
54                       END IF;
55                 END;
56             END IF;
57             PARALLEL-END;
58        MASK:  END;
59        COMPARE:  PROCESSOR;
60              CASE OPCODE;
61                  IN TIMER-VALUES[6] PARALLEL-BEGIN;
62                     IF DOCNUMX < DOCNUMY THEN
62                IN TIMER-VALUES[7] BEGIN;
63                          NEED-XINPUT <- 1;
64                          INPUT-X <- 0;
65                       END;
```

```
 66                    END IF;
 67                    IF DOCNUMX = DOCNUMY  THEN
 67                    IN TIMER-VALUES[8] BEGIN;
 68                          DOCNUMZ <- DOCNUMX;
 69                           OUTPUT-Z;
 70                                  NEED-XINPUT <- 1;
 71                          NEED-YINPUT <-1;
 72                          INPUT-X <- 0;
 73                          INPUT-Y <- 0;
 74                          ZOUTPUT-FLAG <- 1;
 75                       END;
 76                     END IF;
 77                    IF DOCNUMX > DOCNUMY THEN
 77                     IN TIMER-VALUES[9] BEGIN;
 78                         NEED-YINPUT <- 1;
 79                        INPUT-Y <- 0;
 80                        END;
 81                    END IF;
 82                PARALLEL-END;
 83                 IN TIMER-VALUES[10] PARALLEL-BEGIN;
 84                    IF DOCNUMX < DOCNUMY THEN
 84                      IN TIMER-VALUES[11] BEGIN;
 85                          DOCNUMZ <- DOCNUMX;
 86                          OUTPUT-Z;
 87                      NEED-XINPUT <- 1;
 88                         INPUT-X <- 0;
 89                          ZOUTPUT-FLAG <- 1;
 90                        END;
 91                    END IF;
 92                    IF DOCNUMX = DOCNUMY THEN
 92                       IN TIMER-VALUES[12] BEGIN;
 93                          DOCNUMZ <- DOCNUMX;
 94                    OUTPUT-Z;
 95                          NEED-XINPUT <- 1;
 96                         NEED-YINPUT <- 1;
 97                          INPUT-X <- 0;
 98                          INPUT-Y <- 0;
 99                          ZOUTPUT-FLAG <- 1;
100                       END;
101                    END IF;
```

```
102                   IF DOCNUMX > DOCNUMY THEN
102                  IN TIMER-VALUES[13] BEGIN;
103                         DOCNUMZ <- DOCNUMY;
104                         OUTPUT-Z;
105                        NEED-YINPUT <- 1;
106                         INPUT-Y <- 0;
107                         ZOUTPUT-FLAG <- 1;
108                       END;
109                   END IF;
110                PARALLEL-END;
111                 OP-NOTSIM;
112                 OP-NOTSIM;
113                 OP-NOTSIM;
114                 OP-NOTSIM;
115                 OP-NOTSIM;
116                 OP-NOTSIM;
117              END CASE;
118       COMPARE:   END;
119            OUTPUT-Z:   PROCESSOR;
120                   BEGIN;
121               MEM[Z-REG] <- ZOUTPUT-REG1;
122               Z-REG <- Z-REG + 1;
123               END;
124            OUTPUT-Z: END;
125             PC <- 0;
126          DO FOREVER; BEGIN;
128               IF STEP-FLAG
128                   THEN BEGIN;
129                       OP-STEP;
130                   END;
131               END IF;
132               IR <- MEM[PC];
133                X-REG <- MEM[PC+1];
134                Y-REG <- MEM[PC+2];
135                LISTSIZE-XREG <- MEM[PC+3];
136                LISTSIZE-YREG <- MEM[PC+4];
137                Z-REG <- MEM[PC+5];
138                PC <- PC+6 ;
139                NEED-INPUT <- 1;
140                NEED-XINPUT <- 1;
```

```
141                NEED-YINPUT <- 1;
142                DO UNTIL LISTSIZE-FLAG;
143                   BEGIN;
144                     ZOUTPUT-FLAG <- 0;
145                       IF STEP-FLAG
145                       THEN BEGIN;
146                       OP-STEP;
147                      END;
148                    END IF;
149                      X-HOLD1 <- X-IN;
150                       Y-HOLD1 <- Y-IN;
151                     DO UNTIL ZOUTPUT-FLAG;
152
152                      PARALLEL-BEGIN;
153                      IF NEED-INPUT
153                        THEN BEGIN;
154                          DO UNTIL INPUT-X AND INPUT-Y;
155                             BEGIN;
156                               INPUT-FETCH;
157                               MASK;
158                             END;
159                             NEED-COMPARE <- 1;
160                           END;
161                       END IF;
162                       IF NEED-COMPARE
162                         THEN BEGIN;
163                           COMPARE;
164                          END;
165                       END IF;
166                      PARALLEL-END;
167                END;
168                      END;
169          P-MERGE:   END;
```

MODIFICATIONS

BIBLIOGRAPHY

(AFSC78) Using the Air Force Command Computer Network (AFSCNET), April 1978.

(AGRA76) Agrawala, A.K. and Rauscher, T.C., Foundations of Micro-programming, ACM Monograph Series, Academic Press, 1976.

(ARPA76) ARPANET Resource Handbook, NIC Document Number 39335, December, 1976.

(BAGL76) Bagley, J.D., "Micro-programmable Virtual Machines", Computer, February, 1976, Vol. 9.

(BANE79) Banerjee F., Hsiao, D., and Kamnar, K.,"DBC-A Database Computer for Very Large Databases", IEEE Transactions on Computers, June 1979.

(BATC68) Batcher, K.E., "Sorting Networks and Their Applications", Spring Joint Computer Conference, Pg.307-314.

(BEAC) Beach, E.J. and Mercurio, J., "Emulation Capabilities of a Micro-programmable Multi-Processor System", U.S. Army, Fort Monmouth, New Jersey.

(BERR74) Berra, P. Bruce, "Some Problems in Associative Processor Applications to Data Base Management", AFIPS Conference Proceedings, Vol. 43, 1974, pp.1-5.

(BERR78) Berra, P. Bruce, "Recent Developments in Data Base Management and Information Retrieval Hardware Architecture", Proceedings of Comp Sac 78, November, 1978.

(BERR79) Berra, P. Bruce and Oliver E., "The Role of Associative Array Processors in Data Base Machine Architecture", Computer, March, 1979.

(BLOC78) Block, E. and Galage, D., "Component Progress: Its Effect on High-Speed Computer Architecture and Machine Organization", Computer, April, 1978.

(BOEH73) Boehm, B. W., "Keynote Address: The High Cost of Software", TRW Systems Group, TRW-SS-73-08, September, 1973.

(BOLI77) Boli, B., Weinberg, A. and Zolotow, N., NLS User Training Guide, Standford Research Center, ARI/ARC Journal Catalog Number 29174, April 1977.

(BURK) Burkhard, W. A., "Flexible Computer Architectures for Research and Development", Computer Science Department, University of California.

(CANA74) Canaday, R.H., Harrison, R.D., Ivie, E.L., Ryder, J.L., and Wher, L.A., "A Back-end Computer for Data Base Management", Communications of ACM, Vol. 17, No. 10, October 1974, pp.575-582.

(CAPR78) Capraro, G.R., "A Data Base Management Modeling Technique and Special Function Hardware", PHD.Dissertation, Syracuse University, February 1978.

(CAVA79) Cavano, J.P. and McCall, J.A., "A Framework for the Measurement of Software Quality", Proceedings of Software Quality Assurance Workshop, San Diego, California, November, 1978.

(CHAM79) Champine, G.A., "Current Trends in Data Base Systems", Computer, May 1979.

(CHU74) Chu, Yaohan, "Why Do We Need Computer Hardware Description Languages", Computer, December, 1974.

(CLAR79) Clark, N.B. and Troutman, M.A., "The System Architecture Evaluation Facility - An Emulation Facility at Rome Air Development Center", AFIPS Conference Proceedings, NCC, June 1979.

(COPE73) Copeland, G.P., Lipovski, G. J. and Su, S.Y.W., "The Architecture of CASSM: A Cellular System for Non-Numeric Processing", Proceedings of First Annual Symposium on Computer Architecture, December 1973, pp. 121-128.

(CROC77) Crocker, S. D., "State Deltas: A Formalism for Representing Segments of Computation,"PHD. Dissertation, ISI/RR-77-61, October 1977.

(DEFI74) DeFiore, C.R., and Berra, P. Bruce, "A Quantitative Analysis of the Utilization of Associative Memories in Data Base Management", IEEE Transactions,Vol.23, Number 2, February 1974, pp. 121-123.

(DEMC75) Demco, J.C., and Marsland, T.A., "A Complete PDP-11 Emulation", Technical Report 75-13, Computing Science Department, University of Alberta, Edmonton, Canada, August, 1975.

(DEWI79) DeWitt, D.J., "Direct - A Multiprocessor Organization for Supporting Relational Data Base Management Systems", IEEE Transactions on Computers, June 1979.

(DIEB) Diebold, E. K. and Wilhelm, "Emulation Utilizing a General Purpose Microprogrammable Processor", Philadelphia Software Development Center.

(DIET74) Dietmeyer, D.L., "Introduction DDL", Computer, December 1974, pp. 34-38.

(DONO77) Donovan, J.J. and Madnick, S.E., "Institutional and Ad Hoc DSS and Their Effective Use", Data Base, Volume 8, Number 3, December 1978.

(FELD73) Feldman, J.D., and Reeman, O.A., "RADCAP An Operational Parallel Processing Facility", Proceedings of 1973 Sagamore Computer Conference on Parallel Processing, August 1973.

(FINF79) Finfer, M., Fellows, J., Casey, D., "Software Debugging Methodology", RADC-TR-79-57, April 1979, Vol I, AD# A069 539; Vol II, AD# A069 540; Vol III, AD# A069, 541.

(FLIN77) Flink, C.W., "EASY - The Design and Implementation of an Intermediate Language Machine", Naval Surface Weapons Center, NSWC/DL TR-3765, October 77.

(FOST70) Foster, C.C., Computer Architecture, Van Nostrand Reinhold Company, 1970.

(FREE79) Freeman, H.A., Data Base Computer Research, Contract N00014-78-C-0487, November 1979.

(GAGI72) Gagliardi, V.O. and Goldberg, R.P., "Virtualizable Architectures", Proceedings ACM AIAA International Computing Symposium, Venice, Italy, 1972.

(GAGI77) Gagliardi, V.O., Jain, R.K., "Research on Virtual Machines", RADC-TR-77-57, May 77, AD# A040 466.

(GALL78) Gallenson, L., Cooperband, A., and Goldberg, J., "The PRIM System: An Alternative Architecture for Emulator Development and Use", ISI TM-78-7, January 1978, AD# A053 154.

(GOLD73) Goldberg, R.P., "Architecture Principles for Virtual Computer Systems", ESD-TR-73-105, February 1973.

(GOLD74) Goldberg, R.P. and Schwenk, H., "Benefits of Virtual Machine Techniques for INPUT/OUTPUT, Infotech State of Report 22, England, September 1974.

(HILL74) Hill, Federick J., "Introducing AHPL", Computer, December 1974, pp. 28-30.

(HOLL76) Hollaar, L.A., "An Architecture for Efficient Combining of Linearly Ordered Lists", Second Workshop on Computer Architecture for Non-Numeric Processing, Gainesville, Florida, 1976.

(HOLL76) Hollaar, L.A., and Stellhorn, "A Specialized Architecture for Textual Information Retrieval", Technical Report UNIVCDCS-R-74-637, Department of Computer Science, University of Illinois.

(HOLL79) Hollaar, L.A.,"Test Retrieval Computers", Computer, March 1979, Volume 12, Number 3, pp. 40-50.

(HOLL79) Hollaar, L.A., "A Design for a List Merging Network", IEEE Transactions on Computers, June 1979.

(HSIA77) Hsiao, D.K. and Madnick, S.E.,"Data Base Machine Architecture in the Context of Information Technology Evaluation", Proc. Third Int. Conference on Very Large Data Bases, ACM, N.Y., 1977.

(KARL75) Karlowsky, I.,Leilich, H.O., and Ziedler, H.C.H., "Content Addressing in Data Bases by Special Peripheral Hardware: A Proposal Called "Search Processor", Infomatch Farhbirichte 4, Computer Architecture Workshop of the Gesellschaft Fur Informatch, Erlangin, May 1975,pp.113-131.

(KOEH73) Koehr, G. I., Connolly, J.T., Rhymer, P.P., Girken, B.L., Sahr, E.V., Data Management Systems Catalog, Mitre Technical Report - 139, Mitre Corp., January 1973.

(LANC77) Lancaster, D., TTL Cookbook, Howard W. Sams and Co., Inc., 1977

(LAND78) Land, T. and Nahouraii, E., "A Two-Level Architecture for a Large Data Base", Second Workshop on Computer Architecture for Non-Numeric Processing", Gainesville, Florida, 1976.

(LEVE) Leventhal, L.A. and Walsh, W.C., "Memory Technology", Engineering and Technology Department, Grossmont College, California.

(LIPO77) Lipovski, G.J., "Hardware Description Languages: Voices from the Tower of Babel", Computer, 1977.

(LIPO78) Lipovski, G.J. and Dotg, K.L., "Developments and Directions in Computer Architecture", Computer, August 1978.

(LIUZ78) Liuzzi, R.A. and Berra, P.B., "A Data Base Machine Design and Evaluation Facility", Proceedings of Comp Sac 78, Computer Software and Applications Conference, November 1978.

(LOVE73) Love, H.H., "An Efficient Associative Processor Using Bulk Storage", Proceedings of 1973 Sagamore Computer Conference on Parallel Processing, pp. 103-112.

(LOWE76) Lowenthal, E. I., "The Back-end (Data Base) Computer", Part I and II, Averback (Data Management) Series 24-01-04 and 24-01-05, 1976.

(M68075) "M6800 Systems Reference and Data Sheets", Motorola Semiconductor Products, Inc., May 1975.

(MADN78) Madnick, S.E., "Infoplex: A New Concept in Data Base Management Technology", Proceedings of the Third International Conference on Very Large Data Bases, October 1978.

(MADN78) Madnick, S.E. and Donovan, J., "Use of Virtual Machines in the Development of Decision Support Systems", RADC-TR-79-187, July 1979, AD# A073 748.

(MALL75) Mallach,E.G., "Emulator Architecture", Computer 8, 5 August 1975, pp.24-32.

(MALL76) Mallach, E.G., "On the Relationship Between Virtual Machines and Emulators", Honeywell Information Systems.

(MANN76) Manno, M.M.,Computer System Architecture, Prentice-Hall, Inc., 1976.

(MCCA77) McCall, J. A., Richards P.K., and Walters, G.F.,"Factors in Software Quality", RADC-TR-77-369, Vols.I-III, November 1977, AD#s, Vol I, A049 014; Vol II, A049 015; Vol III, A049 055.

(MECL74) Semiconductor Data Libray, MECL Integrated Circuits, Motorola Semiconductor Products, Inc. 1974.

(MOLD73) Molder, R., "An Implementation of a Data Base Management System on an Associative Processor", AFIPS Conference Proceedings, Vol. 42, 1974, pp.171-176.

(MUKH79) Mukhopadhyay, A., "Hardware Algorithms for Non-Numeric Computation", IEEE Transactions on Computers, June 1979.

(NANO76) Nanodata Corporation, QM-1 Reference Manual, March 1976, Williamsville, New York.

(NAUG75) Naughton, J.J. and Smith, R.L., "Structured Programming Series:, Volumes I-XIII, RADC-TR-74-300, July 1975, AD#'s, Vol I, A016 771; Vol II, A018 046; Vol III, A013 255; Vol IV, A015 794; Vol V, A003 339; Vol VI, A007 796; Vol VII, A008 639; Vol VIII, A016 414; Vol IX, A016415, Vol X, A008 861; Vol XI, A016 416; Vol XII, A026 947; Vol XIII, A020 858.

(NORM78) Norman, F., "MULTICS-QM-1 Conversion Program", January 1978.

(NSW75) NSW Computer Connection, Rome Air Development Center, Contract F30 602-75-C-0915, September, 1975.

(OLIV79) Oliver, E.J., "RELACS, An Associative Computer Architecture to Support a Relational Data Model", Phd. Dissertation, Syracuse University, June 1979.

(PARS79) Parsley, B.L., Lehtman, H. G., and Kahn,S., "On-Line Programmer's Management System", RADC-TR-79-205, August 1975, AD# A074 468.

(PHIL78) Phillips, D.E., Brown, J.M., Kellerman, J.P., "MADMAN on H6000, RADC-TR-78-8, June 1978.

(PITC77) Pitchumani, R. and Stabler, E., " An Evaluation of SMITE as a Hardware Description Language", Syracuse University Interim Report- "Studies for a Digital Systems Design Support Facility", March 1977.

(PRES77) Press, B., "SMITE Reference Manual", RADC-TR-77-364, November 1977, AD# A049 038.

(RADC78) Data Base Management System Strategy on Associative Array Processor/Mass Memory (AAP/MM) RADC Contract F30602-78-C-0147, May 1978.

(RAMA) Rammoorthy, C.V., Turner, J. L., and Wah, B.H., "A Design of a Faster Sorting Associative Memory", Computer Science Division, Division of Electrical Engineering, University of California at Berkeley.

(ROBE76) Roberts, D.C., "A Special Computer Architecture for High-Speed Text Searching", Second Workshop on Computer Architecture for Non-Numeric Processing", Gainesville, Florida, 1976.

(ROSI72) Rosin, R.F., Frieder, G., and Eckhouse, R.H., "An Environment for Research in Microprogramming and Emulation", Communications of ACM, August 1972, Volume 15, Number 8.

(SCHT77) Advanced Micro Devices - Shottkey and Low Power Schottky Data Book, 1977.

(SCHU75) Schuster, S.A., Ozkarahan,E.A., and Smith,K.C., "A Virtual Memory System for a Relational Associative Processor", TR CSRG-64, University of Toronto, December 1976.

(SCHU79) Schuster,S.A.,Nguyen,H.B.,Ozkarahan,E.A.,and Smith,K.C., "RAP.2 - An Associative Processor for Data Bases and It's Applications",IEEE Transactions on Computers,June 79.

(SEVC79) Sevcek,K.C. and Klawe,M.M., "Operational Analysis Versus Stockatic Modeling of Computer Systems",Proc. Computer Science and Statistics; 12th Annual Symposium on the Interface, May 1979, Waterloo, Canada.

(SIEW74) Siewiorek, D.P., "Introducing ISP", Computer, December 1974, pp. 39-41.

(SING76) Singhania, A.K. and Berra, P.B., "A Multiple Associative Organization for Pipelining a Directory to a Very Large Data Base", Spring Computer Conference 76 Digest of Papers, 1976, pp. 109-112.

(STEL77) Stellhorn, W.E., "An Inverted File Processor for Information Retrieval", IEEE Transactions on Computers, Volume C-26, December 1977.

(STWE77) Stewart, J.H., "LOGAL: a CHDL for Logic Design and Synthesis of Computers", Computer, June 1977, pp.18-26.

(STIL71) Stillman, N.J., DeFiore, C., Berra, P.B., "Associative Processing of Line Drawings", AFIPS Conference Proceedings, SJCC, Volume 38, May 1971.

(SU79) Su, S.Y.W., Nguyen, L.H., Eman, A. and Lipovski, G.J., "The Architectural Features and Implementation Techniques of the Multicell CASSM, IEEE Transactions on Computers, June 1979.

(TRW77) TRW Defense and Space Group, SMITE Reference Manual, RADC-TR-77-364, November 1977.

(TRW79) TRW Defense and Space Group, Advanced SMITE Reference Manual, Contract F30602-78-C-0016, CDRL 007, February 1980, RADC-TR-80-66.

(VEMU80) Vemuri, V., Liuzzi, R.A., Cavano, J.P., Berra, P.B., "Issues in the Performance Evaluation of Data Base Machine Designs", 1980 Computer Architecture for Non-Numeric Processing Conference, March 1980.

(VEST78) Vestal, S.C., Krocok, T.K., Schwenk,H.S., and Levy, A., "H6180 Virtual Machine Monitor Performance Analysis", RADC-TR-78-251, December 1978, AD# A065 087.

(VEST79) Vestal,S.C., Liuzzi, R.A., Schwenk, H., and Levy, A., "Performance of a Virtual Machine Monitor", Proceedings of Virtual Machines Workshop, Munich, Germany, March 1979.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.